1982-12

Interactive microcomputer control system modeling and realization using a database for automatic programming

Johnsrud, Roger Francis

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTERACTIVE MICROCOMPUTER CONTROL SYSTEM

MODELING AND REALIZATION USING A DATABASE

FOR AUTOMATIC PROGRAMMING

by

Roger Francis Johnsrud

December 1982

Thesis Advisor: A. Gerba Jr.

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Interactive Microcomputer Control System Modeling and Realization Using A Database For Automatic Programming.	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, December 1982
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Roger F. Johnsrud	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: December 1982
	13. NUMBER OF PAGES: 117
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Control Systems	Modeling	Simulation
Microcomputers	Database	Automatic Programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A program for control system modeling, simulation, and realization useable by the novice programmer was developed for interactive use on a microcomputer. The program uses two software packages; PL/1-8Ø from Digital Research and DBASE II from Ashton Tate.

The paper covers a complete example for modeling and simulation of a minimum time response ripple free controller for a $1/S^2$ Plant. The program uses two sample rates, one for modeling the computer and the other for modeling

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the plant. Variable delays due to computer computation are included and corrected semi-transparent to the control designer.

This thesis covers two different fields of interest, one for the experienced control system designer yet a novice programmer, and the other for the novice control designer yet an experienced computer programmer. Key programming concepts include using a business database to automatically program a control problem requiring a scientific language.

Approved for public release; distribution unlimited.

Interactive Microcomputer Control System Modeling and Realization Using a Database for Automatic Programming

by

Roger Francis Johnsrud
Lieutenant, United States Navy
B.S.E.E., University of Washington, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1982

ABSTRACT

A program for control system modeling, simulation, and realization useable by the novice programmer was developed for interactive use on a microcomputer. The program uses two software packages; PL/1-80 from Digital Research and DBASE II from Ashton Tate.

The paper covers a complete example for modeling and simulation of a minimum time response ripple free controller for a $1/S^2$ Plant. The program uses two sample rates, one for modeling the computer and the other for modeling the plant. Variable delays due to computer computation are included and corrected semi-transparent to the control designer.

This thesis covers two different fields of interest, one for the experienced control system designer yet a novice programmer, and the other for the novice control designer yet an experienced computer programmer. Key programming concepts include using a business database to automatically program a control problem requiring a scientific language.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. CONCEPT

The intent of this thesis was to develop a computer program that an experienced controls engineer could use to help him model, simulate, and realize control systems with the convenience of a desktop microcomputer. The control engineer will not need to be an experienced computer programmer. In order to accomplish the above requirements it is necessary that the program be easy to use, effectively requiring that the program write the computer code itself with little help from the control designer. This is commonly referred to as automatic programming. In order for the program to be a useable tool for the controls designer the system must also produce a good model of the system that is being designed. Obviously all control systems cannot be considered in this type of project. The approach taken in this thesis was to use control systems that have error signals designed to be equal to zero or differ from zero by a constant. Other guidelines of this project include: the equations must be simple difference equations, the system must allow for calculation delays within the computer and these delays must be corrected for automatically or at least allow for the possibility that the designer can correct for the delays, and the program must be able to model the plant

between the sample periods of the computer which basically means there are two sample periods, one for the computer and one for the plant. Since the plant cannot be programmed as analog it must be discretized. Using a faster sample rate for the plant allows simpler equations yet maintains a good representation for the plant. Additionally, the program must be able to model analog to digital and digital to analog conversions. After the model has been designed and the simulation run, the results that are available from the program must be easy to use and easy to interpret by the controls engineer. In order to meet all of these requirements, most of the program development itself must be transparent to the controls engineer. The approach taken was for the control engineer to provide only equations, function names, and function relationships by providing the names of other functions used as inputs for each equation.

B. ASSUMPTIONS AND REQUIREMENTS

The assumptions and requirements are stated below. First, as mentioned above, the program must be relatively easy to use. The program must have high precision math capability since it was designed to be used with control systems. Speed of compiling and operation was not considered critical at this point since microprocessors are becoming smaller and faster which will solve any realization problems in the near future. A major consideration is that once the simulations have been run and the results are acceptable to the designer

then the simulation equations should be easily converted into realization equations. Basically this means the simulation equations in the computer should be easily converted into hardware equations that can be used to control a real plant. In order to accomplish this task it was decided that the program must develop source code that could be compiled by a high level language compiler for either computer simulation or hardware implementation. This approach makes the program transportable and hardware independent.

C. IMPLEMENTATION

This project was implemented entirely on microprocessor based systems to show that it is feasible and practical to use these small systems for major development. The present system is slightly constrained due to the length of compile time. All programs were developed using a Z80 microprocessor; however, the code is written in high level source languages therefore the programs will also run on the simpler 8080 based microcomputers or on the more complex sixteen bit microprocessors such as the 8086.

The languages chosen were PL/1-80 from Digital Research because of the math and hardware capabilities of the language allowing for control of analog to digital and other processes inside the computer, and DBASE II from Ashton Tate which allows easy and high level manipulation of databases and file systems. One of the major points of this thesis is to manipulate functions and functional relationships as

databases. Once the relationships have been developed, the database program writes PL/1 source code from the database information. The PL/1 source code is then compiled by the PL/1 compiler and the simulation run on the resident hardware.

The present programs and the test example in this report provide simulation only and send the results in floating point format to a disk file. The floating point file can then be converted to display the information in tabular or graphical form such as shown at the back of this report. The information can be displayed in a variety of forms such as on display consoles, printers, plotters, or graphic terminals. The real power for design is in the use of graphical equipment to provide the control engineer with the entire picture of the system under test. The Hewlet Packard 9872B plotter and 2647A graphics terminal and the Intercolor 8363 graphics terminal were used to show that the data produced was basically machine independent. The code can be compiled, run, and displayed on a variety of different machines.

II. GENERAL DISCUSSION

A. CONTROL ENGINEER INFORMATION

The control system engineer does not need to know how the computer programs work in order to use the system. It is assumed that the control engineer is a novice programmer with some knowledge of the CP/M operating system and limited background in program writing; however, it is assumed he is a knowledgeable control designer. This means he is fairly well versed in taking control systems and producing discrete system equations from the continuous control equations. These equations are designed and optimized by whatever means are available either by hand or on a mainframe computer. (Microcomputers will soon be able to do major system development equations.) Once the equations are developed the designer enters them into a microcomputer and tests the simulations. If the simulations are acceptable then it should be an easy matter to convert the simulation code for the controller design into development code for actual implementation into hardware.

For ease of use the programs are menu driven and use simple cursor arrows or commands for pointing at selections. Additionally very basic information is needed such as the name of functions and equations in a simple format the computer can use. This procedure is accomplished by using the

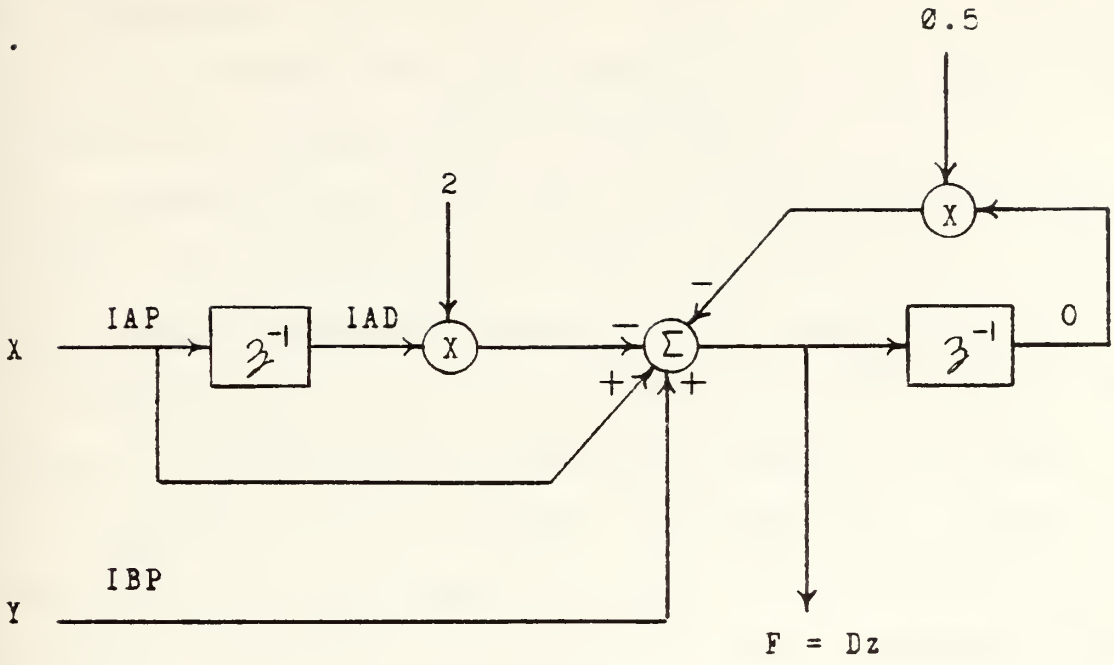
database program which only allows a user to enter information in very specific fields of very specific type. The type of information required from the designer is to add or delete equations, to change initial conditions and to provide function names.

There is an initial learning curve with this program but an attempt has been made to keep it as short as possible. There are some peculiarities put in as a requirement to make interfacing between the human and the computer possible. Some examples include names such as INPUT:A or INPUT:B that indicate which function the designer is talking about to the computer. As an example INPUT:A will associate a function in the computer with the function name the designer has selected such as INPUT:A = Un. Another very abstract name is A1P2D. The user is expected to enter either 1 if he wants the PRESENT value of the function named under INPUT:A or 2 if he wants the DELAYED value of the function named under INPUT:A. If the designer will be using both the present and past values then enter 1 to get the present value and the past value will also be available. The strange requirement for A1P2D is provided only to allow much faster programs when the code is implemented in hardware and the designer only needs past or delayed values of a function. Two other codes are IAP and IAD for INPUT A PRESENT and INPUT A DELAYED. These two codes are provided to keep the equations needed for the functions short. Another variable is "O" which is used to

define the delayed or previous time sample of the functions own OUTPUT. There are similar definitions for up to five inputs INPUT:A through INPUT:E for each function. A function can be defined in three ways; "F" means the function is an equation that the computer finds a register to hold the result in, "AD(x)" means the value is to be placed in one of eight analog to digital registers, and "DA(x)" means the value is to be placed in one of eight digital to analog registers. An example equation and simple block diagram is shown in Figure 1.

Other variables used include "T" for the discrete sample period and "T1" for the sample period of the plant. If the value of T is normalized and T1 has the same value as T then the programs will run faster in the realization of the code since these values effectively disappear from the equations. It should be noted that all functions are done in floating point so the simpler the equations the faster the code. The example problem at the end of this paper normalizes T but still includes it in the equations to show how it could show up in the function. Finally the designer is allowed to specify end criteria and intervals for calculating the plant relative to the computer itself.

After the designer has entered the equations into the database and is satisfied that all function relationships are correct then all that is required is to enter the correct menu selection that compiles the code. The computer program



NOTES:

Assume by previous definition that INPUT:A = X, INPUT:B = Y, FUNCNAME = Dz, and the below equation is desired:

D(k) = X(k) - (2 * X(k-1)) + Y(k) - (0.5 * D(k-1))

Making substitutions:

F = Dz = D(k), IAP = X = X(k), IAD = X(k-1), IBP = Y = Y(k), and O = OUTPUT DELAYED = D(k-1) leads to the equation ---

F = IAP - (2 * IAD) + IBP - (0.5 * O)

Notice that the above function only has two inputs, X and Y

Figure 1

Simple Difference Equation Demonstrating System Variables

will now write the necessary source code from the database and subsequently invoke the PL/1 compiler which will turn the code into a program that will run on any of the previously discussed CP/M based systems. The executable file produced is called ANSWER.COM and is run on a CP/M based system by typing "ANSWER" in response to the operating system input prompt. During program ANSWER.COM execution the results will be sent to a floating point file called DATA.FLT. Also, while the program is running the intermediate results are printed on the display device for program troubleshooting should the results not be as expected. The file DATA.FLT is not printable; however, an example utility program called PRINT.PLI is provided that changes the floating point file into a printable text file. This program is executed by typing PRINT FILE where FILE is the name of the designer provided text file the designer wants the results placed into. With further programming development the floating point data can be converted into any format needed by different devices such as plotters as shown by the plot results in Appendix C.

The information for understanding the database program operation is provided in Chapter V. This section is not necessary for understanding at the design level but it does explain variables that will be needed in equation development. Additionally the example problem will be helpful to the design engineer. When the designer is ready to use the

program all that is needed is to enter "DBASE MENU" to the CP/M operating system prompt. As previously stated the designer will need some familiarity with CP/M before the program can be of use.

B. PROGRAMMER INFORMATION

This section covers the experience necessary for program maintenance and further development. These programs would be worked on and coded by an individual that has a general control system background; however, he must have substantial experience with computer programming. The type of computer experience needed pertains to operating systems, event counters, and automatic programming. The programmer must have a good knowledge of both PL/1-80 and IBASE II. Additionally it would be necessary to know microcomputer hardware implementations such as analog to digital converters and sample and holds. These devices are simulated in the computer programs but further useful extensions would involve implementing the code into realized hardware. The information required for the programmer to understand the programs is contained in the remainder of this thesis. The majority of this thesis is the computer code required to implement a design environment for a novice programmer through the use of automatic programming.

III. SYSTEM IDIOSYNCRASIES

Major problems encountered in this thesis had to do with programming and efficiency. The major problem is that there is no one programming language that contains all the features necessary to implement the entire project. DBASE II was chosen because of the relational database characteristics even though it has very limited math or hardware capabilities. The requirements for reasonable math is the reason PL/1-80 was chosen and though some database systems are avaliable for PL/1 they do not have all the features of DBASE II. A major deficiency of DBASE II is that it will not allow the printing of a semicolon. The semicolon is considered a command character in DBASE II and therefore reserved. This can be considered a major bug for a database system, especially one that was designed for business applications. It is foolish to believe business will not want to include a semicolon someplace in one of their transactions. Ashton Tate was notified and they replied it was not worth trying to fix. This presents a problem since the database program was used to write the source code in PL/1 and PL/1 requires a semicolon at the end of every statement. The problem was worked around by having DBASE II write a backslash every place a semicolon was needed in the PL/1 source text. Next a PL/1 preprocessor called REFORM.PLI

was used to change all backslashes to semicolons prior to invoking the PL/1 compiler. This takes a large amount of the compile time which could have been easily corrected by a reasonable response from Ashton Tate. PL/1 also has some problems; however, Digital Research was more than happy to try to overcome deficiencies. There are several problems in their floating point routines due mainly to conversion rules which can cause strange effects in the results. Most floating point functions such as the exponential function tend to convert the arguments to fixed before calculation and this in effect truncates the arguments. As an example if one desires to raise an exponent to a power and that power is a function such as X/Y then both X and Y and the result get truncated. If X/Y is a fraction less than 1.Ø the effect is to truncate the result to Ø.Ø and the result of the exponent to a fractional power always ends up equal to 1.Ø. The way around this problem is to convert X and Y to CHARACTERS before doing the division. This is only part of the problem. Even if X and Y origionally were floating point and were converted to characters the result of the exponent evaluation is fixed decimal. Now this result must be converted by anotner function called FTC in order to get the result back to the system in floating point. An example of this shows up in the MONITOR.PLI program for the function called CF. This does cause some limitation on the type of equations that the designer can use without knowing some peculiarities of the

designer can use without knowing some peculiarities of the system.

For the sake of program efficiency, as far as development goes, all variables or functions the designer uses are considered floating point. This makes programming easier with higher precision; however, the code takes longer to execute even though the execution time was not considered critical in this implementation. Using floating point for all functions introduces some inaccurate realizations when dealing with analog to digital and digital to analog converters. This is not a major problem since these devices will have greater accuracy in the future or the devices can be modeled more accurately in future designs.

All results presently produced by this system are placed on floppy disks inside the computer. Writing to the disk and displaying the results on a terminal are an obvious slowdown in system operation. These inefficiencies do not show up in a realization of the control system since the controller will be self contained computer code communicating only with the converters. Additionally the code will be faster since it will not be necessary to simulate the plant.

Presently the realization of the code is not implemented. The software code needed for converters and the actual monitor needed for the realized hardware is not developed. This is not a major problem and should be relatively straight forward for a computer programmer with general hardware

experience. The actual control program will be identical and will only need to link into the new converter programs. In addition the simulation display is not fully implemented. To make this project a powerful tool the simulation results should be presented in graphical form to the designer similar to the plots at the end of this paper. If properly displayed, when the designer changes one part of the system all effects throughout the system should be dynamically evident. The advantages of a completely contained hardware, software and display system are obvious.

The time required to compile the programs is definitely a function of the computer hardware. Most newer Z80 based single user systems can compile the code in about fifteen minutes. 8080 based systems, older Z80 based systems, and multiuser systems that share the central processor will require up to thirty minutes to compile. Recall the compile time is not critical as mentioned earlier and the realization code will run in the order of milliseconds. As a final note the programs would run much faster if only one language and no preprocessor were needed.

Another area that requires discussion is the theory involved in the design of the correction factor. Development of a general prediction factor due to delays caused by conversions and computer calculations is covered very briefly in this paper. The practical results of a predictor that is used with the example problem is shown in the graphs at the

end of this paper; however, how the results were obtained and what they really mean is not fully investigated. Hopefully, this will be pursued and discussed in a following paper. The predictor is more a control engineers problem and needs to be investigated extensively to determine the optimum predictor for all possible delays. Further discussion is provided in the example problem.

IV. FUTURE EXTENSIONS

After a careful investigation of the types of problems encountered with various hardware and software interfaces, it becomes very evident that a sixteen bit or larger microprocessor that offers high speed calculation integrated into a complete graphic and plotting machine will be a definite boost to the realization of future work. Additionally a math processor integrated circuit that can operate concurrently with the central processor will be of invaluable importance. Once you place these hardware items along with good converter circuits into one unit it will no longer be necessary to simulate in one machine and realize in another. Once the simulation code works all that will be needed will be to specify executing the realization code on the same machine if connectors are available on the machine for communicating with the plant. Once the system operates correctly the hardware would be reduced to a smaller special purpose system; however, the power of using a microprocessor for the controller cannot be overlooked. If one provides the capability of a full blown microcomputer in the control loop even though all processes are not needed, the system could be sent to a distant changing environment and if the control equations needed to be changed due to unforseen circumstances it could be done remotely. By using the above method it would

be possible to reconfigure and realize a new system dynamically which cannot be done by a physical one time hardware implementation.

Another requirement is the software system needed for this type of development. The problems of integrating two completely different software packages is evident in this paper. An ideal software package would be a scientific relational database that has the capability to control the computer hardware and provide self modifying code or automatic programming for it's own operating system. This type of language would make the programming more efficient, ease the implementation, and provide a friendlier environment for future expansion. This type of software package is realizable under the new concurrent languages being developed such as ADA (product of the Department of Defence). ADA has not been proven to work in microcomputers since it is such an extensive language; however, several companies are showing it to be feasible in the newer computers such as the 68000 microprocessor from Motorola. The 68000 is a thirty two bit processor with a sixteen bit external bus. Motorola is presently designing a version of the 68000 to have a thirty two bit external bus.

It is recommended that future systems contain processors such as the 68000 and math processors such as Intel's 8087 or similar math processors from other companies such as Texas Instruments. Additionally high speed high precision analog to

digital and digital to analog converters should be used. The type of converters used should contain their own multiple input sample and hold circuits. This would insure capturing all inputs at the same time (versus time multiplexing) making the control realizations more practical for real time operations.

V. DESCRIPTION OF DATABASE ALGORITHM

A. GENERAL DISCUSSION (Refer to Appendix A)

This section covers how the database turns the functions into PL/1-80 code. It covers each subroutine and there are hints on how the control designer uses the system. The easiest way for the control designer to use this thesis is to briefly read this material, go through the example problem, and then proceed directly into the program. In general a database is a program that contains related information. Normally these programs are used in business applications where it is desired to keep track of an individuals name, address, or account information; however, databases can be used to store other information. In this program implementation the database contains the function name, the type of function such as an analog function, names of other functions that are used by this function as inputs, the equations for the functions, the relationship between functions, and if the function will be saved for later recall and plotting.

The program has the ability to develop new or recall old system models that have been previously developed. Once the program is entered the model can be changed by adding or deleting transfer functions, change relationships, or change the equations. Additionally the models initial conditions can

be changed. If only the initial conditions are changed the program will compile much faster since each function will not need to be recompiled. After the system model is developed the database proceeds to write PL/1 "include" files that are later included into standard PL/1 modules developed for this project.

The features of the LBASE II program are the high level relational database language and the ability of the program to write text files onto floppy disk. The disadvantages include a limited math capability and it does not provide the ability to interface with the microcomputer hardware. A severe drawback in the program is that it cannot produce a semicolon in any of the text files that it writes.

B. LEVEL #1 (MAJOR SUBROUTINES AND PROCEDURES)

1. MENU.CMD - initial menu selection

a. Initializes all variables and the database.

b. Enters a CASE statement to select a new problem, old problem, save the present problem, display present function names, change only the initial conditions of the present problem, or quit the program.

c. See Figures 15A and 15B in Appendix A.

2. CMENU.CMD - second level menu selection

a. Develops the control model database.

b. Enters a CASE statement to edit, add, delete, recall deleted functions, or change initial conditions.

c. Calls CONTROL.CMD.

d. See Figures 16A and 16B in Appendix A.

3. CONTROL.CMD - database control program

a. Calls procedures DCLMON.CMD, PROC.CMD, DATA.CMD, and FORM.CMD that develop the PL/1 "include" source code.

b. Ends the database program and invokes the PL/1 programs.

4. DCLMON.CMD - monitor declaration program

a. Develops DCLMON.TXT for the MONITOR.PLI program.

b. Calls MONDCL.CMD.

5. MONDCL.CMD - develops AA.TXT, AD.TXT, DD.TXT, and DA.TXT include functions for MONITOR.PLI.

6. PROC.CMD - Develops DCLFUN.XXX, INPUT.XXX, FUNC.XXX, and OUT.XXX for each function developed by the control engineer to be used in the appropriate function AA.PLI, AD.PLI, DD.PLI, or DA.PLI.

7. DATA.CMD - data declaration procedure

a. Develops DATAINFO.TXT to be used in PRINT.PLI

b. DATAINFO.TXT contains valuable format information pertaining to the DATA.FLT file.

c. Calls DATA1.CMD - additional data format file

8. FORM.CMD - PL/1 command execution program

a. Develops submit file FORM.SUB used to control the order of PL/1 program compiling.

b. Issues commands to reformat files containing a backslash into files with a semicolon.

c. Calls DCLFUN.CMD - function name setup for REFORM

d. Calls REFORM.CMD - general format for PL/1 compiling.

9. SHORT.CMD - short compile program

a. Develops initial conditions for DCLMON.TXT and MONITOR.PLI.

b. Issues commands to control the compiling of only MONITOR.PLI by developing a submit file SFORM.CMI.

c. Calls SFORM.CMD - short format for PL/1 compiling similar to FORM.SUB called SFORM.SUB.

C. LEVEL #2 (PROGRAM DESCRIPTION)

Since the majority of the code is in higher level languages, and is designed in modules, it is already in algorithmic form. It is assumed that the individual needing to understand the code will be familiar with references 1 through 4; therefore, an extensive discussion of the computer code will not be provided. The example problem will cover all the features of the program.

The basic structure of the database includes a memory file that contains the value of initial variables and a database file that contains the functions and their relationships to each other. The entire database program is involved with manipulating and recalling information from the two above mentioned files.

The initial memory file is shown in Figure 2 and is contained in file INIT.MEM. This file is provided so that

initial variables called will not produce errors in the database program. The main variables of concern are:

1. T - discrete sample period,
2. T1 - analog sample period,
3. DLIMIT - end criteria for the digital simulation,
4. ALIMIT - number of analog samples for each digital sample.
5. DELAY - delays due to calculation and conversion expressed as the number of analog counts after the sample period T.

Unseen system variables include:

1. ACTION - users next menu selection,
2. PARAM - holds database program name,
3. CN - test variable,
4. FN - test variable,
5. TEMPOUT - temporary variable,
6. ELEMENTS - data array position for a variable,
7. IND - index counter,
8. FUN - string of characters that is used to uniquely define a function filetype,
9. DATA1 - number of data array elements saved.

The basic structure for all functions is shown in Figure 3 and is contained in file INIT.DBF. The initial values of the database are shown at the bottom of the Figure 3. The only items the control designer sees are listed under the NAME column of Figure 3. The user must enter values for the

```
ACTION      (C)  Ø
PARAM       (C)  INIT
CN          (C)  INIT
FN          (C)  FUNCTION
T           (N)   1.ØØØ
DLIMIT      (N)    1Ø
ALIMIT      (N)    5
TEMPOUT     (C)  ØØØ.ØØØØ
ELEMEMTS    (N)   Ø
IND         (N)   Ø
FUN         (C)  ABCLEFGHIJKLMNOPQRSTUVWXYZ
DELAY       (N)   Ø
T1          (N)   1.ØØØ
DATA1       (N)   Ø
```

Figure 2

Variables Used for Initialization

```
STRUCTURE FOR FILE:  INIT.DBF
NUMBER OF RECORDS:   00001
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001      FUNCNAME     C    008
002      FUNCTYPE     C    002
003      NR:INPUTS    C    001
004      INPUT:A      C    008
005      A1P2L        C    001
006      INPUT:B      C    008
007      B1P2L        C    001
008      INPUT:C      C    008
009      C1P2L        C    001
010      INPUT:D      C    008
011      D1P2L        C    001
012      INPUT:E      C    008
013      E1P2L        C    001
014      FK           N    008    004
015      FKL          N    008    004
016      EQUATION     C    050
017      SAVEDATA     L    001
018      DATANAME     C    005
```

```
* Below is a list of the values used in the initialization.
* These values correspond directly to the structure above.

00001  NEW FUNC
       AD
       0
       ONE
       1
       TWO
       1
       THREE
       1
       FOUR
       1
       FIVE
       1
       0.0000
       0.0000
       AD(1)=0
       .F.
       AD(1)
```

Figure 3

Initial Database Values

variables under NAME of the correct type and size specified. The definitions for the variables are given below:

1. FUNCNAME - function name used to identify this function initialized to "NEW FUNC".

2. FUNCTYPE - describes this function's type as;

   a. AA - analog,

   b. AD - analog to digital,

   c. DD - digital,

   d. DA - digital to analog.

3. NR:INPUTS - This is the number of other functions this function uses for inputs. AL(x) and DA(x) registers are not counted as functions.

4. INPUT:A - This is a variable for name of function used as an input and is shown at the bottom of Figure 3 as "ONE". This value is referred to in the equations as IAP or IAD for INPUT A PRESENT or INPUT A DELAYED.

5. A1P2D - This is a code to tell the computer the designer is talking about the "A" input and he wants either a "1" for the PRESENT value or "2" for the DELAYED value. Entering a "1" will make both the present and delayed values available but the program will run slower than if only the delayed value is needed.

6. INPUT:B - This variable is identical to 4 above except this is the "B" input and has an initial name of "TWO" as shown at the bottom of Figure 3. INPUT:C, INPUT:D, and

INPUT:E are the same as above which allows for up to five inputs.

7. B1P2D - This variable is the same as 5 above except it is for the "B" input function. C1P2D, D1P2D, and E1P2D all have the same function except they are used with their respective inputs.

8. FK - This function is the present value of the function named under FUNCNAME above.

9. FKD - this is the delayed or previous sample period value of the function named in FUNCNAME (ie. FK delayed).

10. EQUATION - this is the equation for the function named in FUNCNAME above. The equation takes three forms;

F = --------- this is a function,

AD(x) = ----- value for an analog to digital,

DA(x) = ----- value for a digital to analog.

11. SAVEDATA - logical value indicating if the user wants to save the value of this function for tabulation.

12. DATANAME - if it is decided to save the function then what type function is it;

F - the function itself,

AD(x) - one of the eight analog to digital,

DA(x) - one of the eight digital to analog.

It should be noted that functions are of three types. If the computer finds a location for the function or it is not hardware dependent then the function is described as "F". If the function is one of the other two types "AD(x)" or "DA x)"

then they represent physical fixed registers within the computer. If the latter two types are used as arguments in an equation they are not listed as inputs under NR:INPUTS for the function. Since they are physical registers they are called by their name (ie. AD(5)) and not by IAP or IAD.

D. LOGICAL STRUCTURE

The logical structure discussed will be a general outline of program flow. A complete discussion is covered in the example problem. Initially the control designer enters the program by responding "IBASE MENU" to the CP/M operating system prompt. This will place the program under control of MENU.CMD which initializes the program and enters a CASE statement that provides a menu selection shown in Figure 15A. At this point the designer usually starts a new problem or recalls an old problem that modifications will be made to. Notice that in order to save a problem the user must select from this menu option. In general the next choice of the designer is to enter the modification phase either selecting to change only the initial conditions of an old problem or selecting to make major changes. If major or new changes are chosen the program transfers control to CMENU.CMD where the designer is allowed to change the function database by menu selections shown in Figure 16A. The choices are controlled by the CASE statement in CMENU.CMD. Most selections are single command entries followed by a "RETURN" from the keyboard. Some commands will operate without a "RETURN" especially if

the user has exceeded a field in one of the database records. The fields are marked on the display with a ":" at each end of the field. On some terminals the arrow keys may not always work for positioning the cursor in a field desired. In this case use "CTRL E" (control key pressed at the same time as the "E" key) to move the cursor up a field. Use "CTRL X to move down a field and "CTRL C" to move down a complete record.

Once the user is sure the database contains all "correct" functions and relationships he should return to the "MAIN PROGRAM" if he wants to save the information otherwise any changes will be lost during compiling. The program could be improved by adding a question in the CONTROL.CMD procedure asking the user if he wants to save the information before compiling. It takes more than fifteen minutes to compile the program so a few extra minutes checking for correctness before compiling can be well worth the effort.

Once the option to compile the program is taken, control passes to CONTROL.CMD. This is the main control procedure that causes the automatic programming by calling all necessary procedures required to write the PL/1 source code from the database. A complete discussion of this procedure is contained in the example problem. After the PL/1 source code is written, a control program is written on how to compile the PL/1 code. Finally the database transfers control to the control program it just wrote, called FORM.SUB, by exiting

the database and submitting FORM.SUB to the CP/M operating system. At this point the PL/1 programs are compiled and executed.

VI. DESCRIPTION OF PL/1 ALGORITHM

A. GENERAL DISCUSSION (Refer to Appendix B)

This section describes the PL/1 programs and how they interact to provide simulation of a computer controller for an analog plant. There are five basic procedures consisting of a monitor that insures all functions are calculated at least once during each sample period of the computer, an analog procedure for describing analog calculations, a digital to analog procedure and analog to digital procedure for describing conversions, and a digital procedure for describing computer calculations. There is only one monitor but there are as many of the other procedures as needed to describe the entire system model for each function. As an example if there are two analog to digital equations in the problem then there would be two analog to digital procedures, one for each equation. The example problem will demonstrate the function relationships. The basic procedures require information for each function and that is provided by the control engineer through the database program in the form of include files. Once the program is ready and compiled automatically by the database program a result file called ANSWER.COM is produced. The problem is run by typing "ANSWER" to the CP/M operating system prompt. As the program is running intermediate results are displayed on the display

device for troubleshooting should the results not be as expected. After the program runs it produces a file DATA.FLT containing floating point data. This file can be translated into a text file by typing PRINT FILE where FILE is a name of a file provided by the user that he wants the results placed into.

PL/1 was chosen as the operating system language because of several features. First it is a common high level language that is transportable to many of the present microcomputers. The intent of this project was to keep the programs as machine independent as possible and using PL/1 source code is a good way to do this. PL/1 also has very good math capabilities. Since the type of problems being solved are engineering in nature it was necessary to have a high level math language. Additionally, the ability for PL/1 to communicate with the computer hardware is a definite asset if the programs are to be realized into hardware. Several other languages were looked at such as PASCAL but PASCAL does not have the ability to talk to the hardware. The 'C' language was also looked at but it lacks the math capability. As stated in Chapter IV.; hopefully, ADA will overcome all the difficulties by combining all the needed features into one language.

PL/1-80 has some problems. There is a definite problem with conversion factors as was discussed in Chapter III. The main problems have to do with conversions between floating

point and other forms. The original approach was to use only floating point values and therefore not need conversions but it was later discovered that several functions do conversions regardless of starting and ending types. This is a problem and cannot be overlooked by the designer if he has problems with results. If equations requiring complex functions are used the designer will need to become familiar with PL/1.

B. LEVEL #1 (MAJOR SUBROUTINES AND PROCEDURES)

1. MONITOR.PLI - main control loop program

a. Causes iteration of all functions.

b. Insures all functions are called in proper order.

c. Saves data for later display and analysis.

2. AA.PLI - basic format for all analog functions

a. Calls all functions used as inputs.

b. Calculates and returns it's own present or past values to all functions that call it.

c. Insures synchronization of calculations with the system clocks through event counts.

3. AD.PLI - basic format for all analog to digital functions and performs identical functions to those shown in 2a, 2b, and 2c above.

4. DD.PLI - basic format for all digital functions and performs identical functions to those shown in 2 above.

5. DA.PLI - basic format for all digital to analog functions and performs identical functions to those above.

6. REFORM.PLI - corrects all improperly formed PL/1 "include" files by changing the backslash produced by the BDASE II program into semicolons.

7. PRINT.PLI - utility program showing how to convert the floating point file DATA.FLT produced by the main program into a different printable text file for display on the system display.

C. LEVEL #2 (PROGRAM DESCRIPTION)

Since the majority of the code is in higher level languages, and is designed in modules, it is already in algorithmic form. It is assumed that the individual needing to understand the code will be familiar with references 1 through 4; therefore, an extensive discussion of the computer code will not be provided. The example problem will cover all the features of the program.

The basic structure of the program uses a monitor that sequentially calls all functions within the computer once during each digital clock. The system clock is used as an event counter to provide synchronization for each function. The functions are developed by the control engineer through the database program and each functional relationship indicating which functions call each other must be properly provided. This is taken care of when the designer states which functions are inputs to each other. A function can have its output connected to any number of functions but a

function can only have five inputs not counting analog to digital, digital to analog, and it's own delayed output. The function definition allows for very complex multiple input/output relationships within any single equation. A method for correcting for computation and other delays is provided in the correction factor CF in the monitor. This is a factor provided for the example problem; however, it is felt that this equation will work for any control system of the type requiring a steady state error signal of zero. The CF is provided but can be eliminated simply by not referencing it in any of the equations. The programs provided are for simulation only but should be easily realized into control hardware since the design equations will not change.

D. LOGICAL STRUCTURE

The program contains a number of procedures that are basic to all problems. These were discussed above. Additionally each problem has unique features that are provided by the control designer through the database programs. These features are coded into PL/1 "include" files and read in at compile time. Once the programs are coded all information as to which functions are connected together will be provided. The monitor is a DO WHILE LOOP that calls all analog to digital and digital functions once each count of the clock. At each count of the digital clock (DCLOCK) any called function will perform it's own equation evaluation if it has not already been calculated during this clock and

return the correct value to the invoking function. When the monitor calls a function it does not use the value returned. Additionally there is a DO WHILE LOOP inside the main monitor loop used to call the analog functions the number of times specified by the designer to occur between each digital sample period. This is the analog clock (ACLOCK). Also during the analog clock when the ACLOCK is equal to the DELAY specified by the designer the digital to analog function is invoked. When a function is invoked, the first thing that it does is check to see if it has any inputs. If there are any inputs then the input functions are called to insure they are the correct value before the equation is calculated. Even though the monitor is a sequential process invoking each function, once a function is called it will invoke all of the input functions needed for equation calculation. Some functions could be called many times during each clock depending on how many other functions use it's output. Once each DCLOCK the monitor outputs values that have been requested by the designer to save into a file.

The first procedures called by the monitor are all digital to analog functions located in a file written by the database program called "ad.txt". The functions take the form dummy_ptr = FUNCTION_NAME where FUNCTION_NAME is the name of the function and the file FUNCTION_NAME contains the necessary information to represent the function. The example problem has a more detailed discussion. The first thing the

digital to analog functions do is call the analog functions that are used as inputs. If these analog functions have other analog inputs they call their inputs before calculating their output results. This process will continue until the analog functions have no inputs or their inputs come from a digital to analog register that will contain a value for a correct point in time. Similar processes occur for files "dd.txt", "da.txt", and "aa.txt" representing digital, digital to analog, and analog functions respectively. The order of calling is controlled initially by the monitor and is extremely important for all processes to obtain the correct results. Direct coding of the processes would be much more efficient but it must be remembered that these processes must work no matter how the designer initially defined the problem. This method of problem solution has merits even for the experienced programmer. Even though the code is longer than direct programming this type of problem will typically require one or two hundred files introducing the possibility for syntax and other programming errors.

The four main functions aa.pli, ad.pli, dd.pli, and da.pli all have the same general structure. Basically the function is called by the monitor or another function, the function calls any inputs, and then returns the correct value of the equation at the time the equation was invoked. This arrangement places a process on the stack through the PL/1 calling routine until all inputs are available eliminating

the need for developing a special process manager. Using this approach makes the programs easily adapted to concurrent processors and languages such as ADA. The conversion routines ad.pli and da.pli do not have a provision for using their delayed values since they would not be available in real hardware. This forces a true representation of the hardware and makes the designer provide intermediate holding functions if delayed values are needed. Additionally the actual code for hardware will be easier to implement at a later date. The other functions dd.pli and aa.pli have provision to return a previously calculated value if only the delayed value is desired. This allows the code to run much faster and provides more powerful equations within a single function. All four processes allow for up to five function inputs and the procedure knows how to call those inputs by using the variable "nr_inputs" to determine to which label in the function to jump. Basically all functions the designer implements are written into one of the above four procedures. As an example if the designer specifies a digital function called DIGIT, the database will write the include files and then compile the dd.pli procedure into a file DD.REL. Next the compiled file DD.REL will be copied into DIGIT.REL which will now contain all the necessary information for the function DIGIT. Additionally the necessary information such as the name DIGIT will be included into the monitor so that it knows to call the function once each clock period.

Before any of the PL/1 programs can be compiled, all the text files written by DBASE II that contain a backslash must be converted to semicolons. This is accomplished by the procedure REFORM.PLI. This is a simple program that reads in a file, translates the character, and outputs to a new file. The files reading from and too are controlled by the submit files FORM.SUB or SFORM.SUB written by the DBASE II program.

The programs compile into a program called ANSWER.COM under the control of the database program. ANSWER.COM is the file needed to run the problem. The designer only needs to place this file on a CP/M based system and type ANSWER in response to the operating system prompt. This will run the program and send the intermediate results to the display and send the floating point data into a file called DATA.FLT.

Once the program has been run it may be desired to place the floating point data into a different form for display and analysis. This project can be greatly improved by providing the results on a graphical display device. Both tabular and graphical results are supplied at the end of this paper and the reader can determine for himself which is the best method. One utility program provided is called PRINT PLI. This program reads in the floating point data by using the information contained in the file DATAINFO.TXT as to how the data was stored and then sends the information into a text file specified by the user. This text file can then be printed if desired.

VII. EXAMPLE

A. INTRODUCTION

This part of the paper will make reference to all aspects of the project and provides the easiest method for understanding the entire paper. It should be noted that the descriptions as to how the data is entered and displayed is somewhat misrepresented since the entire project was designed for an interactive environment. The actual data entry at runtime is much more formal with data fields that allow only specific entry in specific areas of very specific type. This is not a constraint but an aid. As an example if the user is to replay if he wants to save the function results he can only answer with "T, F, Y, or N" for "true, false. yes, or no". The answers can be in upper or lower case and if an improper input is made such as "7" the computer will BEEP and wait for a correct response. Additionally some information is supplied in reverse video or highlighted. In the event that a color system is used such as the INTERCOLOR the information will be in different colors. The only way to fully appreciate the aids for the designer is to run the program. In order to provide a realistic test for this program the sample problem will include a plant that is at the limit of stability and is initially designed to be controlled by a bang bang controller. The process will include a control loop that

feeds back on itself which provides a definite headache to computer programmers because of the synchronization problems.

B. PROBLEM SETUP

Assume we have a continuous $1/S^2$ plant and it is desired to produce a minimum time response ripple free controller L(k) for a unit step input. Referring to Figure 4A below, L(k) is the computer transfer function, T is the sample rate of the input, D is the delay from input sample at time T to output sample including all computation and conversion delays, and the clamp is included since the computer output is held fixed between samples. Assuming that the clamp has a

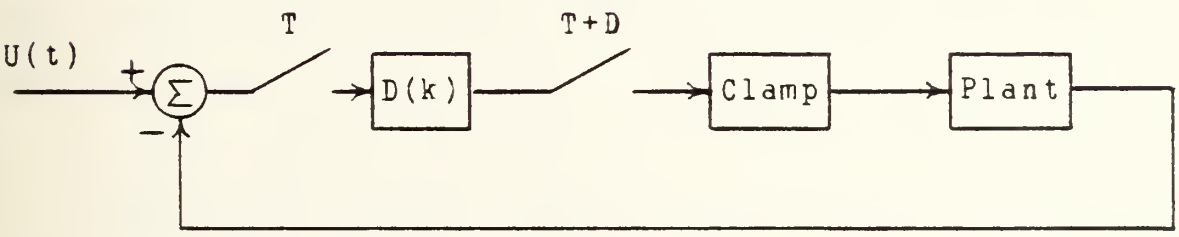


Figure 4A

Computer and Plant Model

very short conversion delay compared to all other delays in the digital system, the output of the clamp is for all practical purposes the same as the input to the clamp. This has been demonstrated by many authors [Ref. 5]. The next initial assumption will be to ignore any delays due to conversions and calculations in the computer (ie set D = Ø). This will allow for simple design equations yet the gross error in reasoning will be easily corrected for later. (It should be noted that most control systems ignore the delays due to computer computation since the delays are usually insignificant compared to the sample rate.)

By letting X2(t) = the plant output, X1(t) = d/dt[X2(t)], and letting the input to the plant be E2(t) by ignoring the effects of the clamp and computation delay, a set of discrete difference equations can be developed describing the plant at the sample instances T. These equations have been developed in the past by many authors [Ref. 6] and are shown below in discrete form:

$$X1(k+1) = X1(k) + T * E2(k) \quad (1)$$

$$X2(k+1) = T * X1(k) + X2(k) - [(T * T) / 2] * E2(k) \quad (2)$$

where;

d/dt[X1(t)] evaluates to X1(k+1)

d/dt[X2(t)] evaluates to X2(k+1)

Now the next task is to control the plant described by the equations in 1 and 2 above according to the design requirements for a minimum time response ripple free

controller. Since this is a second order system it will take two time periods to obtain control. Assume the initial state vector equals zero (ie. X1(Ø) = X2(Ø) = Ø). Since we want the second time state with X1(2) = Ø or no ripple and X2(2) = 1, the input unit step, this leads to the set of equations below after some manipulation:

$$X1(2) = T * E2(Ø) + T * E2(1) = Ø \quad (3)$$

$$X2(2) = [3*(T * T)/2]*E2(Ø) + [(T * T)/2]*E2(1) = 1 \quad (4)$$

Solving equations 3 and 4 above simultaneously leads to:

$$E2(Ø) = 1/(T * T) \quad (5)$$

$$E2(1) = - 1/(T * T) \quad (6)$$

By definition of the Z transform, equations 5 and 6 and the fact that the desired output requires E2(k) = Ø for k > 1 leads to:

$$E2(z) = 1/(T * T) - [1/(T * T)]z^{-1} \quad (7)$$

Now to find the Z transform of E1(k) we start with the problem definition of zero initial conditions and a unit step input.

At k = Ø

E1(Ø) = U(Ø) - X2(Ø)

= 1 - Ø = 1

At k = 1

E1(1) = U(1) - X2(1)

= 1 - [(T * T)/2] * E2(Ø)

= 1 - [(T * T)/2] * 1/(T * T)

= 1 - Ø.5 = Ø.5

At k = 2

E1(2) = U(2) - X2(2)

= 1 - 1 = 0

For K > 1 E1(k) = 0

The above set of equations leads to the Z transform:

$$E1(z) = 1 + 0.5z^{-1} \quad (8)$$

Since D(z) = E2(z) / E1(z), the algorithm for the computer, we can now take the ratio of equations 7 and 8 and cross multiply then transform to find the difference equations needed.

$$[1 + 0.5z^{-1}]*E2(z) = [1/(T * T)] * [1 - z^{-1}]*E1(z) \quad (9)$$

$$E2(k) + 0.5*E2(k-1) = [1/(T * T)]*[E1(k) - E1(k-1)] \quad (10)$$

$$E2(k) = - 0.5*E2(k-1) + [1/(T * T)]*[E1(k)-E1(k-1)] \quad (11)$$

Equation 11 is the one used for the computer simulation and shown in Figure 4B along with the rest of the model. Since it is not allowed to use subscripted or arrays for variable names in this program implementation the following changes have been made in Figure 4B for variables in equations 1, 2, and 11:

EK = E1(k)

E1(k-1) referenced as the delayed value of EK or IAL in the function E2K.

E2K = E2(k)

E2(k-1) referenced as the delayed value of E2K or "0" in the function E2K.

X1N = X1(k) or X1(n) if n different from k

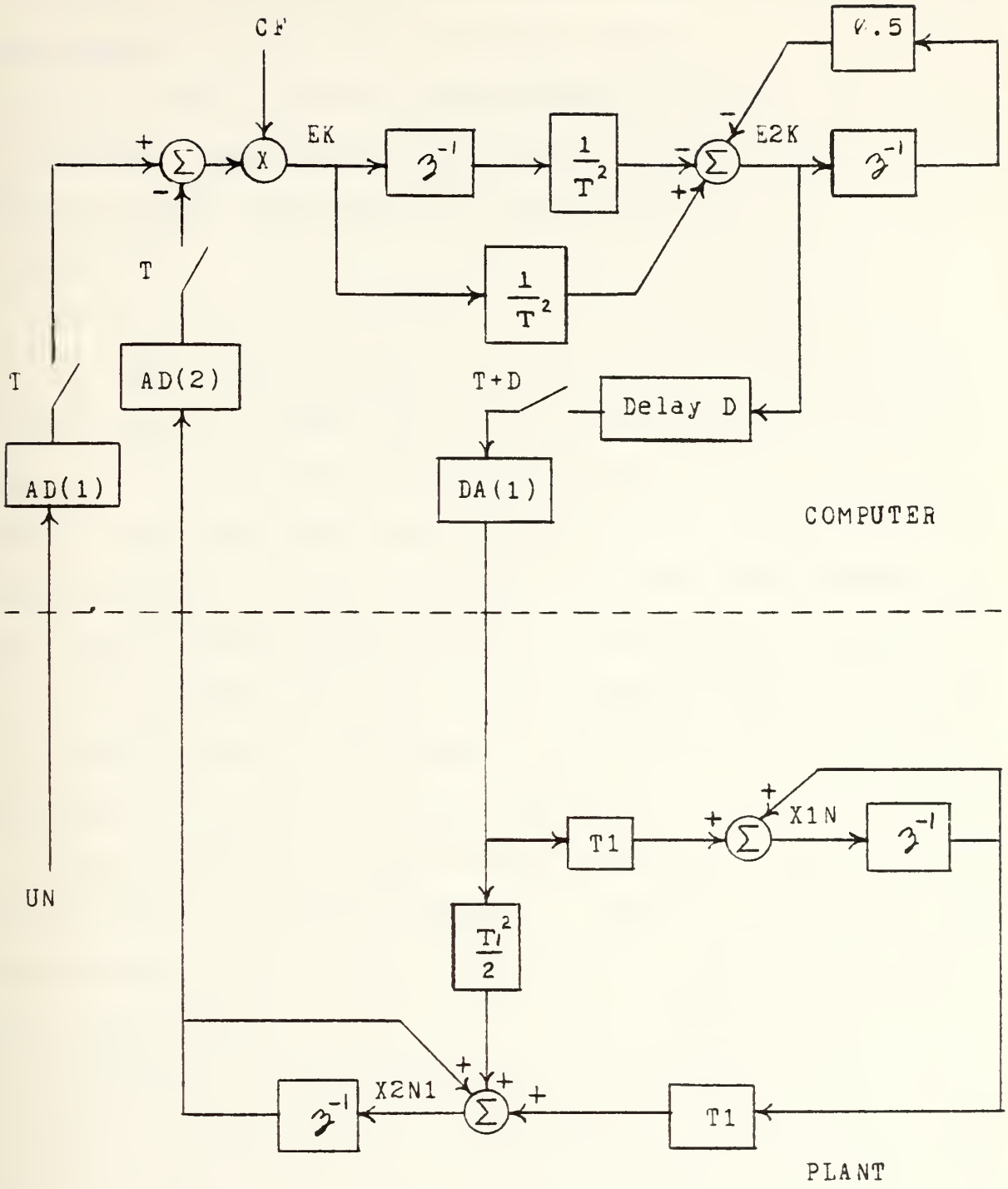


Figure 4B

Test Program Block Diagram

X1(k-1) referenced as the delayed value of X1N or "0" in the function X1N and IAL in function X2N1.

X2N1 = X2(k) or X2(n) if n different from k

X2(k-1) referenced as the delayed value of X2(k) or "0" in the function X2N1 and IAL in function AD2.

UN = U(t)

T1 = [T * i] / j

In the initial design the sample rate of the digital system T and the sample rate of the analog system at discrete points T1 were considered the same(ie. i = j). The only reason for providing the possibility of calculating the analog values at different points is to see what happens to the plant between sample points of the digital system at T. In most problems T and "i" will be normalized to unity and "j" will represent the number of sample points calculated by the plant for each unit sample period T.

Now to account for the combined delays in the discrete system due to analog to digital, digital to analog, and computation delays. This function is called CF or the Correction Factor. The CF in most systems is a major consideration in initial equation development and usually resembles Kalman Filters or predictors. The approach taken for this thesis was that the average designer would not know or could not estimate these delays due to system variables. This means the simulations would not work as expected and needless to say neither would the realizations of the control

algorithms in the real hardware. The attempt was to force the designer to consider the delays in his design and possibly provide a good predictor initially for his use. It was assumed that since this type control system requires an error signal that is zero when in control this would indicate that the signal is zero mean. Since the instantaneous input is random it was assumed the error signal was gaussian and provide a standard exponential predictor to the error signal to correct for delays. The predictor chosen was $CF = e^{-D}$ where e is the natural log and D is the combined delays. The CF is not a complete guess and further work will be presented at a later date. The results of the CF to this particular example can be seen in the plots at the end of this paper. Further research needs to be done to see if this correction works for truncation, roundoff, or even noise in the input signal. The point is that some type of correction needs to be added and this one works very well in this case. If the CF is not wanted the designer only needs to eliminate it from any of his equations.

C. AUTOMATIC PROGRAM (Refer to Appendix A and B)

The designer will initially enter the program by entering "DBASE MENU" and a RETURN to the CP/M operating system prompt. This assumes the disk containing the DBASE programs is in drive "A" and the disk containing the PL/1 programs is in drive "B". The system can be modified to run on a single disk as was done in the original system but the disk must

contain at least one megabyte and should have directory space for at least one hundred fifty files. By using the program setup requirements above the database is entered by menu selection "1" of Figure 15A to start a new program. This will initialize the system and enter the menu selection of Figure 16A. Next if menu selection "5" is entered the initial conditions of Figure 2 can be changed to the ones shown in Figure 5A. This is done by the display menu CINIT of Figure 17A. Note that some parameters of Figure 5A are effected by returning to the main menu and saving the program under the name STRT, others are developed as part of the program process such as ACTION which changes each time a menu selection is made, and others such as DATA1 are developed by the program itself counting the number of items that have been requested to save. As can be seen from Figure 5A, ALIMIT was entered as 10 which means there will be 10 analog calculations for each digital calculation. Since the initial design used the same sample time for both the digital and analog then T1 must be adjusted to 0.1 as discussed previously since T is normalized to 1. After the initial conditions ALIMIT, DLIMIT, DELAY, T, and T1 are entered. the program returns back to CMENU of Figure 16A.

Next assume menu selection "2" is selected and the functions are added. The structure will remain basically the same with only slight changes shown in Figure 5B. The database for the problem will look something like that shown

```
ACTION      (C)  3
PARAM       (C)  STRT
CN          (C)
FN          (C)  FUNCTION
ELEMENTS    (N)   0
DLIMIT      (N)    10
ALIMIT      (N)    10
T1          (N)  0.100
FUN         (C)  ABCDEFGHIJKLMNOPQRSTUVWXYZ
IND         (N)    7
T           (N)   1.000
DELAY       (N)    4
DATA1       (N)    8
```

Figure 5A

Variables Used in STRT Initialization

```
STRUCTURE FOR FILE:  STRT.DBF
NUMBER OF RECORDS:   00008
DATE OF LAST UPDATE: 11/11/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001     FUNCNAME       C    008
002     FUNCTYPE       C    002
003     NR:INPUTS      C    001
004     INPUT:A        C    008
005     A1P2D          C    001
006     INPUT:B        C    008
007     B1P2D          C    001
008     INPUT:C        C    008
009     C1P2D          C    001
010     INPUT:D        C    008
011     D1P2D          C    001
012     INPUT:E        C    008
013     E1P2D          C    001
014     FK             N    008    004
015     FKD            N    008    004
016     EQUATION       C    050
017     SAVEDATA       L    001
018     DATANAME       C    005
```

Figure 5B

Structure for STRT

in Figure 6A - 6D below. Keep in mind that the display looks quite different due to color, highlighting, or reverse video. Also, only one record at a time will be on the screen.

The remainder of the menu programs will not be covered since they are self explanatory. It should be noted that in order to save the problem the designer is working on, he must first return to the main menu and then save the parameters. The program will ask for the name of the file the user wants to save the program in. The example problem was saved in file STRT. If there are questions about the menus look at procedures MENU.CMD and CMENU.CMD in Appendix A. After all functions are entered CORRECTLY, and the program is saved if desired, then enter "Ø" for menu selection in Figure 16A. This will cause the program to enter the automatic programming phase. (Take a break because it will take about fifteen minutes.) If the program fails to compile it will probably be due to improper naming of functions. Function names cannot be arrays. If the results are not as expected, even though the program compiles, then the equations are probably in error or there is a strange conversion in PL/1. If this happens then unfortunately references 2 through 4 must be studied.

Once compiling begins control passes to CONTROL.CMD. The first function called is DCLMON.CMD which writes the monitor declarations DCLMON.TXT for MONITOR.PLI and files AA.TTT, AD.TTT, DD.TTT, and DA.TTT containing the procedure calls for

```
RECORD 00001
    FUNCNAME      :UN       :
    FUNCTYPE      :AA:
    NR:INPUTS     :0:
    INPUT:A       :ONE      :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  1.0000:
    FKD           :  0.0000:
    EQUATION      :F=1.0000                                  :
    SAVEDATA      :T:
    DATANAME      :F    :


RECORD 00002
    FUNCNAME      :EK       :
    FUNCTYPE      :DI:
    NR:INPUTS     :0:
    INPUT:A       :ONE      :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :F=(AD(1)-AI(2))*CF                        :
    SAVEDATA      :T:
    DATANAME      :F    :
```

Figure 6A

Database For Example Problem

```
RECORD 00003
    FUNCNAME      :E2K      :
    FUNCTYPE      :DD:
    NR:INPUTS     :1:
    INPUT:A       :EK       :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :F=(IAP-IAD)/(T*T)-.5*O                :
    SAVEDATA      :T:
    DATANAME      :F    :


RECORD 00004
    FUNCNAME      :X1N      :
    FUNCTYPE      :AA:
    NR:INPUTS     :0:
    INPUT:A       :ONE      :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :F=IA(1)*T1+O                          :
    SAVEDATA      :T:
    DATANAME      :F    :
```

Figure 6B

Database For Example Problem

```
RECORD 00005
    FUNCNAME      :X2N1     :
    FUNCTYPE      :AA:
    NR:INPUTS     :1:
    INPUT:A       :X1N      :
    A1P2D         :2:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :F=DA(1)*T1*T1/2+IAD*T1+O          :
    SAVEDATA      :T:
    DATANAME      :F    :


RECORD 00006
    FUNCNAME      :AD1      :
    FUNCTYPE      :AD:
    NR:INPUTS     :1:
    INPUT:A       :UN       :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :AD(1)=IAP                    :
    SAVEDATA      :T:
    DATANAME      :AD(1):
```

Figure 6C

Database For Example Problem

```
RECORD 00007
    FUNCNAME      :AD2      :
    FUNCTYPE      :AD:
    NR:INPUTS     :1:
    INPUT:A       :X2N1     :
    A1P2D         :2:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :AD(2)=IAD                    :
    SAVEDATA      :T:
    DATANAME      :AD(2):


RECORD 00008
    FUNCNAME      :DA1      :
    FUNCTYPE      :DA:
    NR:INPUTS     :1:
    INPUT:A       :E2K      :
    A1P2D         :1:
    INPUT:B       :TWO      :
    B1P2D         :1:
    INPUT:C       :THREE    :
    C1P2D         :1:
    INPUT:D       :FOUR     :
    D1P2D         :1:
    INPUT:E       :FIVE     :
    E1P2D         :1:
    FK            :  0.0000:
    FKD           :  0.0000:
    EQUATION      :DA(1)=IAP                    :
    SAVEDATA      :T:
    DATANAME      :DA(1):
```

Figure 6D

Database For Example Problem

the monitor. See Figure 7. Next PROC.CMD is invoked for each of the four types of functions. This procedure writes the declarations for each function called DCLFUN.XXX where XXX stands for which function number it is (ie. A, B, C etc. and the last two XX indicate the function type (ie. AA, AD, DD, DA). PROC.CMD also writes the function input files INPUT.XXX, function equation files FUNC.XXX, and the output files OUT.XXX that are used to indicate which element in the data array each function represents. See Figures 8A through 8H. The next function invoked is DATA.CMD. This procedure writes the file DATAINFO.TXT that tells how many functions are stored in the floating point file DATA.FLT and the location of each element. See Figure 9. Next the program passes control to FORM.CMD. This procedure writes the program that tells the computer what to do with all the files that have just been written. Basically it produces a CP/M SUBMIT file that tell the system to reform or correct all files that have a backslash into files containing semicolons, then it tells the system to compile each function and place it under it's own unique function name. After compiling each function it links them all together into one program called ANSWER.COM. See Figure 10. Finally after the database program writes the SUBMIT program it passes control to the program it just wrote and creates the simulation program ANSWER.COM as just mentioned.

```
/* DCLMON.TXT */

    UN        ext entry(fixed(7)) returns(ptr),
    X1N       ext entry(fixed(7)) returns(ptr),
    X2N1      ext entry(fixed(7)) returns(ptr),
    AD1       ext entry(fixed(7)) returns(ptr),
    AD2       ext entry(fixed(7)) returns(ptr),
    DA1       ext entry(fixed(7)) returns(ptr),
    EK        ext entry(fixed(7)) returns(ptr),
    E2K       ext entry(fixed(7)) returns(ptr),
    T float static ext init(  1.0000),
    T1 float static ext init(  0.1000),
    DLIMIT fixed static init(  10),
    ALIMIT float static init(  10),
    delay float static init(   4),
    1 dat ext,
        2 data(  8) dec(15,6)


/* AA.TXT */
    dummy_ptr = UN       (code)\
    dummy_ptr = X1N      (code)\
    dummy_ptr = X2N1     (code)\

/* AD.TXT */
    dummy_ptr = AD1      (code)\
    dummy_ptr = AD2      (code)\

/* DD.TXT */
    dummy_ptr = EK       (code)\
    dummy_ptr = E2K      (code)\

/* DA.TXT */
    dummy_ptr = DA1      (code)\
```

Figure 7

Monitor Declarations and Functions

```
/* DCLFUN.AAA */

UN      :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(0),
ONE       ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(1,1,1,1,1),
F float static init(  1.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.AAA */
input(1) = ONE     \
input(2) = TWO     \
input(3) = THREE   \
input(4) = FOUR    \
input(5) = FIVE    \
/* FUNC.AAA */
F=1.0000
/* OUT.AAA */
data(  1) = ftc(F    )\
put skip list('data(  1) = ',F    )\
```

Figure 8A

Function Declarations and Equations

```
/* DCLFUN.BAA */

X1N     :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(0),
ONE       ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(1,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.BAA */
input(1) = ONE      \
input(2) = TWO      \
input(3) = THREE    \
input(4) = FOUR     \
input(5) = FIVE     \
/* FUNC.BAA */
F=DA(1)*T1+O
/* OUT.BAA */
data(  2) = ftc(F     )\
put skip list('data(  2) = ',F     )\
```

Figure 8B

Function Declarations and Equations

```
/* DCLFUN.CAA */

X2N1    :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(1),
X1N       ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(2,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.CAA */
input(1) = X1N     \
input(2) = TWO     \
input(3) = THREE   \
input(4) = FOUR    \
input(5) = FIVE    \
/* FUNC.CAA */
F=DA(1)*T1*T1/2+IAD*T1+O
/* OUT.CAA */
data(  3) = ftc(F     )\
put skip list('data(  3) = ',F     )\
```

Figure 8C

Function Declarations and Equations

```
/* DCLFUN.AAD */

AD1      :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(1),
UN        ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(1,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.AAD */
input(1) = UN       \
input(2) = TWO      \
input(3) = THREE    \
input(4) = FOUR     \
input(5) = FIVE     \
/* FUNC.AAD */
AD(1)=IAP
/* OUT.AAD */
data(  4) = ftc(AD(1))\
put skip list('data(  4) = ',AD(1))\
```

Figure 8D

Function Declarations and Equations

```
/* LCLFUN.BAD */

AI2     :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(1),
X2N1      ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(2,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* LCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.BAD */
input(1) = X2N1     \
input(2) = TWO      \
input(3) = THREE    \
input(4) = FOUR     \
input(5) = FIVE     \
/* FUNC.BAD */
AD(2)=IAD
/* OUT.BAD */
data(  5) = ftc(AD(2))\
put skip list('data(  5) = ',AD(2))\
```

Figure 8E

Function Declarations and Equations

```
/* DCLFUN.ADD */

EK      :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(0),
ONE       ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
coa(5) fixed(7) static init(1,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.ADD */
input(1) = ONE     \
input(2) = TWO     \
input(3) = THREE   \
input(4) = FOUR    \
input(5) = FIVE    \
/* FUNC.ADD */
F=(AD(1)-AD(2))*CF
/* OUT.ADD */
data(  6) = ftc(F    )\
put skip list('data(  6) = ',F    )\
```

Figure 8F

Function Declarations and Equations

```
/* DCLFUN.BDD */

E2K     :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(1),
EK        ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(1,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.BDD */
input(1) = EK       \
input(2) = TWO      \
input(3) = THREE    \
input(4) = FOUR     \
input(5) = FIVE     \
/* FUNC.BDD */
F=(IAP-IAD)/(T*T)-.5*O
/* OUT.BDD */
data(  7) = ftc(F     )\
put skip list('data(  7) = ',F     )\
```

Figure 8G

Function Declarations and Equations

```
/* DCLFUN.ADA */

LA1      :proc(code) returns(ptr)\
dcl
nr_inputs fixed(7) static init(1),
E2K       ext entry(fixed(7)) returns(ptr),
TWO       ext entry(fixed(7)) returns(ptr),
THREE     ext entry(fixed(7)) returns(ptr),
FOUR      ext entry(fixed(7)) returns(ptr),
FIVE      ext entry(fixed(7)) returns(ptr),
cod(5) fixed(7) static init(1,1,1,1,1),
F float static init(  0.0000),
O float static init(  0.0000)\
dcl
  1 dat ext,
    2 data(  8) dec(15,6)\

/* DCLINP.TXT  - THIS IS PART OF ALL FUNCTIONS AND WAS
                 FIXED FOR THE THESIS CASE OF NUMERIC
                 CONTROL SYSTEMS. */
dcl
   1 IA based(ia_ptr),
     2 IAP float,
     2 IAD float,
   1 IB based(ib_ptr),
     2 IBP float,
     2 IBD float,
   1 IC based(ic_ptr),
     2 ICP float,
     2 ICD float,
   1 ID based(id_ptr),
     2 IDP float,
     2 IDD float,
   1 IE based(ie_ptr),
     2 IEP float,
     2 IED float;
/* INPUT.ADA */
input(1) = E2K      \
input(2) = TWO      \
input(3) = THREE    \
input(4) = FOUR     \
input(5) = FIVE     \
/* FUNC.ADA */
LA(1)=IAP
/* OUT.ADA */
data(  8) = ftc(LA(1))\
put skip list('data(  8) = ',LA(1))\
```

Figure 8H

Function Declarations and Equations

```
/* DATAINFO.TXT */

time1 fixed static init( 100),
nr_data fixed static init(  8),
nr_aa fixed static init( 3),
nr_ad fixed static init( 2),
nr_dd fixed static init( 2),
nr_da fixed static init( 1),
name(  9) char(8) static init(
'UN      ',
'X1N     ',
'X2N1    ',
'AD1     ',
'AD2     ',
'EK      ',
'E2K     ',
'DA1     ' )
```

Figure 9

DATAINFO.TXT

```
/* FORM.SUB */

XSUB
REFORM AA.TXT AA.TTT
REFORM AD.TXT AD.TTT
REFORM DD.TXT DD.TTT
REFORM DA.TXT DA.TTT
PLI MONITOR


REFORM INPUT.AAA INPUT.TTT
REFORM DCLFUN.AAA DCLFUN.TTT
PIP FUNC.TXT=FUNC.AAA
REFORM OUT.AAA OUT.TTT
PLI AA
PIP UN.REL=AA.REL

REFORM INPUT.BAA INPUT.TTT
REFORM DCLFUN.BAA DCLFUN.TTT
PIP FUNC.TXT=FUNC.BAA
REFORM OUT.BAA OUT.TTT
PLI AA
PIP X1N.REL=AA.REL

REFORM INPUT.CAA INPUT.TTT
REFORM DCLFUN.CAA DCLFUN.TTT
PIP FUNC.TXT=FUNC.CAA
REFORM OUT.CAA OUT.TTT
PLI AA
PIP X2N1.REL=AA.REL

REFORM INPUT.AAD INPUT.TTT
REFORM DCLFUN.AAD DCLFUN.TTT
PIP FUNC.TXT=FUNC.AAD
REFORM OUT.AAD OUT.TTT
PLI AD
PIP AD1.REL=AD.REL

REFORM INPUT.BAD INPUT.TTT
REFORM DCLFUN.BAD DCLFUN.TTT
PIP FUNC.TXT=FUNC.BAD
REFORM OUT.BAD OUT.TTT
PLI AD
PIP AD2.REL=AL.REL
```

Figure 10A

FORM.SUB

```
REFORM INPUT.ADD INPUT.TTT
REFORM DCLFUN.ADD DCLFUN.TTT
PIP FUNC.TXT=FUNC.ADD
REFORM OUT.ADD OUT.TTT
PLI DD
PIP EK.REL=DD.REL

REFORM INPUT.BDD INPUT.TTT
REFORM DCLFUN.BDD DCLFUN.TTT
PIP FUNC.TXT=FUNC.BDD
REFORM OUT.BDD OUT.TTT
PLI DD
PIP E2K.REL=DD.REL

REFORM INPUT.ADA INPUT.TTT
REFORM DCLFUN.ADA DCLFUN.TTT
PIP FUNC.TXT=FUNC.ADA
REFORM OUT.ADA OUT.TTT
PLI DA
PIP DA1.REL=DA.REL


PLI PRINT
LINK PRINT

LINK ANSWER=MONITOR ,UN ,X1N ,X2N1 ,AD1 ,AD2 ,DA1 ,EK ,E2K
```

Figure 10B

FORM.SUB

D. TEST RESULTS AND PROGRAM EXECUTION

As mentioned previously, once the xzwgzau ANSWER.COM is developed, all that is necessary is to enter ANSWER to the CP/M operating system prompt. Figure 4B and 9 should be referenced during this discussion. Figure 11 is a partial printing of what will be on the display when the program is executing. The printout is for the sample problem but with the DELAY set to zero and the DLIMIT shortened. The first thing to note is that the functions are not executed in order and some are executed several times during each clock. Time Ø actually runs from DCLOCK = Ø to ACLOCK = 1. By associating the data element number with the order of the function names in Figure 9 the user can determine which function is being calculated or called. As an example the first element returning an answer in Figure 11A is element 4 which equates to AD1 in Figure 9. The next elements are 5, 6, and 7 that equate to AD2, EK, and E2K respectively. An important note is that since there is no delay, function 3 (DA1) or the digital to analog function is calculated or called at time Ø. This will not be the case when discussed later on. The only function that has a value is element 1 or UN. There is no further changes in the data until DCLOCK = 1 which is the same time as ACLOCK = Ø for the second time. At this point the analog to digital conversion has taken place for the first time, the computer does internal calculation, digital to analog is complete since there is no delay, and the analog

plant starts to integrate the signal. This is really the start of the sample since the computer is operating on the initial conditions for the first ten counts of the analog clock. By the first sample period after the initial values (DCLOCK = 2) the controller output changes to -1 and the plant output is at the half way value of the input. By the second sample period after the initial conditions (DCLOCK = 3) the plant output is equal to the forcing function input and the controller output is Ø. It should be noted that in all programs the zero point of the sample period always starts at DCLOCK = Ø so that the initial conditions can be graphed. The above example is a theoretical non realizable simulation of a control system with no calculation delays. The results obtained are identical to classical textbook examples.

Figure 12 is the identical problem discussed with the delay equal to zero but it shows what the display file looks like after the utility program PRINT.PLI is run. The results in Figure 12 are obviously more readable since the function name and final value are not presented until after all calculations are complete.

Figure 13 is a partial printing of the results when using the full example presented above with a forty percent delay. This means there is four ACLOCK counts after the DCLOCK before the digital to analog values are available. The important points to note are that the digital to analog

values (DA1 or function 8) do no change until the analog clock (ACLOCK) is equal to 4 during each digital clock, the overshoot is only about 2.5 percent, and it takes five times as long to reach stability or the input signal. These results are much more representative of a real world problem and the problems that would be encountered with a true hardware implementation.

```
          0 DCLOCK
data(   4) =    0.000000E+00
data(   5) =    0.000000E+00
data(   6) =    0.000000E+00
data(   7) =    0.000000E+00
          0 ACLOCK          0 TIME
data(   8) =    0.000000E+00
data(   1) =    1.000000E+00
data(   2) =    0.000000E+00
data(   3) =    0.000000E+00
          1 ACLOCK          1 TIME
data(   1) =    1.000000E+00
data(   2) =    0.000000E+00
data(   2) =    0.000000E+00
data(   3) =    0.000000E+00


          9 ACLOCK          9 TIME
data(   1) =    1.000000E+00
data(   2) =    0.000000E+00
data(   2) =    0.000000E+00
data(   3) =    0.000000E+00
          1 DCLOCK
data(   1) =    1.000000E+00
data(   4) =    1.000000E+00
data(   3) =    0.000000E+00
data(   5) =    0.000000E+00
data(   6) =    1.000000E+00
data(   6) =    1.000000E+00
data(   7) =    1.000000E+00
          0 ACLOCK         10 TIME
data(   7) =    1.000000E+00
data(   8) =    1.000000E+00
data(   1) =    1.000000E+00
data(   2) =    1.000000E-01
data(   2) =    1.000000E-01
data(   3) =    4.999999E-03
          1 ACLOCK         11 TIME
data(   1) =    1.000000E+00
data(   2) =    2.000000E-01
data(   2) =    2.000000E-01
data(   3) =    1.999999E-02
          2 ACLOCK         12 TIME
data(   1) =    1.000000E+00
data(   2) =    3.000000E-01
data(   2) =    3.000000E-01
data(   3) =    4.499999E-02
```

Figure 11A

ANSWER.COM Runtime Results

```
          9 ACLOCK          19 TIME
data(  1) =    1.000000E+00
data(  2) =    1.000000E+00
data(  2) =    1.000000E+00
data(  3) =    0.500000E+00
          2 DCLOCK
data(  1) =    1.000000E+00
data(  4) =    1.000000E+00
data(  3) =    0.500000E+00
data(  5) =    0.500000E+00
data(  6) =    0.500000E+00
data(  6) =    0.500000E+00
data(  7) =   -1.000000E+00
          0 ACLOCK          20 TIME
data(  7) =   -1.000000E+00
data(  8) =   -1.000000E+00
data(  1) =    1.000000E+00
data(  2) =    0.900000E+00
data(  2) =    0.900000E+00
data(  3) =    0.595000E+00


          9 ACLOCK          29 TIME
data(  1) =    1.000000E+00
data(  2) =    1.490116E-08
data(  2) =    1.490116E-08
data(  3) =    1.000000E+00
          3 DCLOCK
data(  1) =    1.000000E+00
data(  4) =    1.000000E+00
data(  3) =    1.000000E+00
data(  5) =    1.000000E+00
data(  6) =    0.000000E+00
data(  6) =    0.000000E+00
data(  7) =    0.000000E+00
          0 ACLOCK          30 TIME
data(  7) =    0.000000E+00
data(  8) =    0.000000E+00
data(  1) =    1.000000E+00
data(  2) =    1.490116E-08
data(  2) =    1.490116E-08
data(  3) =    1.000000E+00
          1 ACLOCK          31 TIME
data(  1) =    1.000000E+00
data(  2) =    1.490116E-08
data(  2) =    1.490116E-08
data(  3) =    1.000000E+00
```

Figure 11B

ANSWER.COM Runtime Results

```
 TIME=0
UN        =          1.000000
X1N       =          0.000000
X2N1      =          0.000000
AD1       =          0.000000
AD2       =          0.000000
EK        =          0.000000
E2K       =          0.000000
DA1       =          0.000000
 TIME=1
UN        =          1.000000
X1N       =          0.000000
X2N1      =          0.000000
AD1       =          0.000000
AD2       =          0.000000
EK        =          0.000000
E2K       =          0.000000
DA1       =          0.000000


 TIME=9
UN        =          1.000000
X1N       =          0.000000
X2N1      =          0.000000
AD1       =          0.000000
AD2       =          0.000000
EK        =          0.000000
E2K       =          0.000000
DA1       =          0.000000
 TIME=10
UN        =          1.000000
X1N       =          0.100000
X2N1      =          0.004999
AD1       =          1.000000
AD2       =          0.000000
EK        =          1.000000
E2K       =          1.000000
DA1       =          1.000000
 TIME=11
UN        =          1.000000
X1N       =          0.200000
X2N1      =          0.019999
AD1       =          1.000000
AD2       =          0.000000
EK        =          1.000000
E2K       =          1.000000
DA1       =          1.000000
```

Figure 12A

PRINT.PLI With Zero Delay

```
 TIME=19
UN       =          1.000000
X1N      =          1.000000
X2N1     =          0.500000
AD1      =          1.000000
AD2      =          0.000000
EK       =          1.000000
E2K      =          1.000000
DA1      =          1.000000
 TIME=20
UN       =          1.000000
X1N      =          0.900000
X2N1     =          0.595000
AD1      =          1.000000
AD2      =          0.500000
EK       =          0.500000
E2K      =         -1.000000
DA1      =         -1.000000


 TIME=29
UN       =          1.000000
X1N      =          0.000000
X2N1     =          1.000000
AD1      =          1.000000
AD2      =          0.500000
EK       =          0.500000
E2K      =         -1.000000
DA1      =         -1.000000
 TIME=30
UN       =          1.000000
X1N      =          0.000000
X2N1     =          1.000000
AD1      =          1.000000
AD2      =          1.000000
EK       =          0.000000
E2K      =          0.000000
DA1      =          0.000000
 TIME=31
UN       =          1.000000
X1N      =          0.000000
X2N1     =          1.000000
AD1      =          1.000000
AD2      =          1.000000
EK       =          0.000000
E2K      =          0.000000
DA1      =          0.000000
```

Figure 12B

PRINT.PLI With Zero Delay

```
 TIME=0
UN        =            1.000000
X1N       =            0.000000
X2N1      =            0.000000
AD1       =            0.000000
AD2       =            0.000000
EK        =            0.000000
E2K       =            0.000000
DA1       =            0.000000
 TIME=1
UN        =            1.000000
X1N       =            0.000000
X2N1      =            0.000000
AD1       =            0.000000
AD2       =            0.000000
EK        =            0.000000
E2K       =            0.000000
DA1       =            0.000000


 TIME=9
UN        =            1.000000
X1N       =            0.000000
X2N1      =            0.000000
AD1       =            0.000000
AD2       =            0.000000
EK        =            0.000000
E2K       =            0.000000
DA1       =            0.000000
 TIME=10
UN        =            1.000000
X1N       =            0.000000
X2N1      =            0.000000
AD1       =            1.000000
AD2       =            0.000000
EK        =            0.670319
E2K       =            0.670319
DA1       =            0.000000
 TIME=11
UN        =            1.000000
X1N       =            0.000000
X2N1      =            0.000000
AD1       =            1.000000
AD2       =            0.000000
EK        =            0.670319
E2K       =            0.670319
DA1       =            0.000000
```

Figure 13A

PRINT.PLI With Forty Percent Delay

```
 TIME=12
UN       =          1.000000
X1N      =          0.000000
X2N1     =          0.000000
AD1      =          1.000000
AD2      =          0.000000
EK       =          0.670319
E2K      =          0.670319
DA1      =          0.000000
 TIME=13
UN       =          1.000000
X1N      =          0.000000
X2N1     =          0.000000
AD1      =          1.000000
AD2      =          0.000000
EK       =          0.670319
E2K      =          0.670319
DA1      =          0.000000
 TIME=14
UN       =          1.000000
X1N      =          0.067031
X2N1     =          0.003351
AD1      =          1.000000
AD2      =          0.000000
EK       =          0.670319
E2K      =          0.670319
DA1      =          0.670319
 TIME=15
UN       =          1.000000
X1N      =          0.134063
X2N1     =          0.013406
AD1      =          1.000000
AD2      =          0.000000
EK       =          0.670319
E2K      =          0.670319
DA1      =          0.670319


 TIME=49
UN       =          1.000000
X1N      =          0.041242
X2N1     =          1.016266
AD1      =          1.000000
AD2      =          0.922099
EK       =          0.052219
E2K      =         -0.096455
DA1      =         -0.096455
```

Figure 13b

PRINT.PLI With Forty Percent Delay

```
 TIME=50
UN       =          1.000000
X1N      =          0.031597
X2N1     =          1.019908
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.096455
 TIME=51
UN       =          1.000000
X1N      =          0.021951
X2N1     =          1.022585
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.096455
 TIME=52
UN       =          1.000000
X1N      =          0.012306
X2N1     =          1.024298
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.096455
 TIME=53
UN       =          1.000000
X1N      =          0.002660
X2N1     =          1.025047
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.096455
 TIME=54
UN       =          1.000000
X1N      =          0.001170
X2N1     =          1.025238
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.014894
```

Figure 13C

PRINT.PLI With Forty Percent Delay

```
 TIME=55
UN       =          1.000000
X1N      =         -0.000318
X2N1     =          1.025281
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.014894
 TIME=56
UN       =          1.000000
X1N      =         -0.001807
X2N1     =          1.025175
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.014894
 TIME=57
UN       =          1.000000
X1N      =         -0.003297
X2N1     =          1.024919
AD1      =          1.000000
AD2      =          1.016266
EK       =         -0.010903
E2K      =         -0.014894
DA1      =         -0.014894


 TIME=99
UN       =          1.000000
X1N      =         -0.000246
X2N1     =          0.999450
AD1      =          1.000000
AD2      =          1.000391
EK       =         -0.000262
E2K      =          0.001068
DA1      =          0.001068
 TIME=100
UN       =          1.000000
X1N      =         -0.000139
X2N1     =          0.999431
AD1      =          1.000000
AD2      =          0.999450
EK       =          0.000368
E2K      =          0.000096
DA1      =          0.001068
```

Figure 13D

PRINT.PLI With Forty Percent Delay

VIII. CONCLUSIONS

A major point of this thesis was to demonstrate the feasibility of writing a computer program that can be used on a microcomputer by a novice programmer to develop control system models, simulation, and realization. This type of system was not possible in the past due to system memory limitations and unacceptable software programs. The approach taken in this project was to use commercially available high level programming languages to reduce the coding time. As was discussed in this thesis, the languages are not sufficient in themselves and more than one language was needed to complete the project. It is apparent that a language that has the features of a good relational database coupled with extremely good mathematical functions and the ability to communicate with the computer hardware is a necessity for this type of system to be a useful tool.

The importance of recognizing that control systems can be modeled and manipulated in a database cannot be overlooked. By carrying this approach to the limit it is possible to develop a large database of control functions that could be recalled and used like a library. Once the database is formed it could be manipulated by artificial intelligence. The computer itself could be told to develop a control system to match a desired set of waveforms using the information stored

or at least find a best fit solution. Missing parameters could be searched for, or best fit solutions could possibly be provided as easily as using the the present database systems to find phone numbers and names.

It is significant that source code was used for the intermediate language. A special purpose program could have been written that provided direct function calls to specific hardware; however, by using high level language source code the program will run on many different machines just by using their resident compilers. This was demonstrated by transporting the programs to four different machines with different operating environments. By using source code the problem was kept at a high level of abstraction. The programs the computer needed to write were similar to English which helped lessen the confusion while developing more than one hundred files and programs.

A fact that became very apparent early in the program development was the need for a correction factor to take into account the delays in the system due to calculations and conversions. This problem is addressed in many models but it is seldom considered in most computer solutions since the computation time is usually insignificant. It was determined that solving for all possible delays was next to impossible especially when implemented since the hardware itself would be variable. A correction factor is provided that works very well with the example problem. Further investigation needs to

be done using other functions to test the results; however, it is felt that the factor will work with any control system designed to have a zero steady state error signal. The basic approach is to have the designer develop the model assuming there are no delays and then estimate the delays to be applied to the correction factor. If the factor does not work then try to develop one from other experience. If this fails then use benchside engineering, make estimates and watch what the results do.

Appendix C contains several graphs of data covering the example problem. A variety of delays were introduced and the results with and without the correction factor are provided. The problem was initially designed assuming no delay and then corrected for with the correction factor. It should be noted that in all cases, even with delays up to 99 percent, acceptable results were produced within five times the initial sample period design. This is significant since the uncorrected functions had severe problems even with the small delays. When using the correction factor the longer delays corrected better and it is felt that this is due to outputting a control signal closer to the time the input was going to sample the result of the control. If a signal is output and sampled immediately, the probability that the response to that signal is the one observed is higher than if a long delay were encountered possibly resulting in severe data error. Even though the system was designed for a step

input there is evidence that the correction factor works very well for a ramp input. This is also shown in Appendix C.

As a final note the power of a graphical dynamic display for the designer is invaluable. Reading through many pages of data makes it very difficult to see where the significant changes are being made. A real time plot will cut design time by an order of magnitude.

APPENDIX A

DBASE II PROGRAMS

CINIT	DBF	:	CINIT	FMT	:	CINIT	ZIP	:	CINIT	ZPR
CMENU	CMD	:	CMENU	FMT	:	CMENU	ZIP	:	CMENU	ZPR
CONTROL	CMD	:	PARAMS	DBF	:	DATA1	CMD	:	DBASE	COM
DBASEAPP	OVR	:	DBASEBRO	OVR	:	DBASEJOI	OVR	:	DBASEMAI	OVR
DBASEMOD	OVR	:	DBASEMSC	OVR	:	DBASEMSG	COM	:	DBASERPG	OVR
DBASESRT	OVR	:	DBASETTL	OVR	:	DBASEUPD	OVR	:	DBINSTL	COM
PARINDEX	NDX	:	SFORM	SUB	:	DCLFUN	CMD	:	FORM	SUB
FORM1	CMD	:	INIT	DBF	:	INIT	MEM	:	MENU	CMD
MENU	FMT	:	MENU	ZIP	:	MENU	ZPR	:	STRT	DBF
STRT	MEM	:	SUBMIT1	COM	:	ZIP	COM	:	ZIPIN	COM
XSUB	COM	:	SHORT	CMD	:	DCLMON	CMD	:	MONDCL	CMD
PROC	CMD	:	DATA	CMD	:	FORM	CMD	:	REFORM	CMD

Figure 14

Programs Required for Database Operation

```
* MENU.CMD
SET TALK OFF
RESTORE FROM INIT          * See Figure 2
USE INIT                   * See Figure 3
SET FORMAT TO MENU         * See Figure 15A and 15B
DO WHILE T
  READ
  DO CASE
    CASE ACTION = '0'
       ACCEPT 'DO YOU REALLY WANT TO QUIT? Y/N ' TO CN
       IF CN = 'Y'
          QUIT
       ENDIF
    CASE ACTION = '1'
       RESTORE FROM INIT
       USE INIT
       COPY TO PARAMS
       USE PARAMS
       DO CMENU
    CASE ACTION = '2'
       ACCEPT 'ENTER NAME OF OLD PROGRAM FILE  ' TO PARAM
       RESTORE FROM &PARAM
       USE &PARAM
       COPY TO PARAMS
       USE PARAMS
       DO CMENU
    CASE ACTION = '3'
       ACCEPT 'ENTER NAME OF FILE TO SAVE PARAMATERS IN ' TO PARAM
       SAVE TO &PARAM
       COPY TO &PARAM
    CASE ACTION = '4'
       DISP ALL FUNCNAME
       ACCEPT 'PRESS RETURN TO CONTINUE  ' TO CN
    CASE ACTION = '5'
       ACCEPT 'ENTER NAME OF OLD PROGRAM FILE  ' TO PARAM
       RESTORE FROM &PARAM
       USE &PARAM
       COPY TO PARAMS
       USE PARAMS
       DO SHORT
  ENDCASE
ENDDO
```

```
***** file MENU ***

            ENTER Ø TO QUIT
            ENTER 1 TO START A NEW PROBLEM
            ENTER 2 TO RECALL AN OLD PROBLEM
            ENTER 3 TO SAVE PRESENT PARAMATERS
            ENTER 4 TO DISPLAY PRESENT FUNCTIONS
            ENTER 5 TO CHANGE INITIAL CONDITIONS ONLY
                    ( This will allow the program
                      to compile much faster )

                 ENTER ACTION #ACTION
```

Figure 15A

MENU

```
* MENU.FMT
@  7,17 SAY "ENTER Ø TO QUIT"
@  8,17 SAY "ENTER 1 TO START A NEW PROBLEM"
@  9,17 SAY "ENTER 2 TO RECALL AN OLD PROBLEM"
@ 10,17 SAY "ENTER 3 TO SAVE PRESENT PARAMATERS"
@ 11,17 SAY "ENTER 4 TO DISPLAY PRESENT FUNCTIONS"
@ 12,17 SAY "ENTER 5 TO CHANGE INITIAL CONDITIONS ONLY"
@ 13,28 SAY "( This will allow the program"
@ 14,30 SAY "to compile much faster )"
@ 16,23 SAY "ENTER ACTION"
@ 16,36 GET ACTION
```

Figure 15B

MENU Format

```
* CMEMU.CMD
SET FORMAT TO CMENU    * See figure 16A and 16B
DO WHILE T
  READ
  DO CASE
     CASE ACTION = '0'
       PACK
       INDEX ON FUNCTYPE TO PARINDEX
       USE PARAMS INDEX PARINDEX
       DO CONTROL
     CASE ACTION = '1'
       ERASE
       DISP ALL FUNCNAME
       ACCEPT 'ENTER RECORD  NUMBER TO EDIT  ' TO CN
       EDIT &CN
     CASE ACTION = '2'
       APPEND FROM INIT    * See figure 3
       EDIT #
     CASE ACTION = '3'
       ERASE
       DISP ALL FUNCNAME
       ACCEPT 'ENTER RECORD NUMBER TO DELETE  ' TO CN
       DELETE RECORD &CN
     CASE ACTION = '4'
       ERASE
       DISP ALL FUNCNAME
       ACCEPT 'ENTER RECORD NUMBER TO RECALL  ' TO CN
       RECALL RECORD &CN
     CASE ACTION = '5'
       SET FORMAT TO CINIT   * See figure 17A and 17B
       READ
       SET FORMAT TO CMENU   * See figure 16A and 16B
       ACCEPT 'PRESS RETURN TO CONTINUE  ' TO CN
     CASE ACTION = '6'
       SET FORMAT TO MENU    * See figure 15A and 15B
       RETURN
  ENDCASE
ENDDO
RETURN
```

```
***** file CMENU ***

       ENTER Ø TO COMPILE A PROGRAM
       ENTER 1 TO EDIT TRANSFER FUNCTIONS
       ENTER 2 TO ADD A TRANSFER FUNCTION
       ENTER 3 TO DELETE A TRANSFER FUNCTION
       ENTER 4 TO RECALL A DELETED TRANSFER FUNCTION
       ENTER 5 TO CHANGE INITIAL CONDITIONS
       ENTER 6 TO RETURN TO THE MAIN PROGRAM

              ENTER ACTION #ACTION
```

Figure 16A

CMENU

```
* CMENU.FMT
@  3,15 SAY "ENTER Ø TO COMPILE A PROGRAM"
@  4,15 SAY "ENTER 1 TO EDIT TRANSFER FUNCTIONS"
@  5,15 SAY "ENTER 2 TO ADD A TRANSFER FUNCTION"
@  6,15 SAY "ENTER 3 TO DELETE A TRANSFER FUNCTION"
@  7,15 SAY "ENTER 4 TO RECALL A DELETED TRANSFER FUNCTION"
@  8,15 SAY "ENTER 5 TO CHANGE INITIAL CONDITIONS"
@  9,15 SAY "ENTER 6 TO RETURN TO THE MAIN PROGRAM"
@ 12,22 SAY "ENTER ACTION"
@ 12,35 GET ACTION
```

Figure 16B

CMENU Format

```
***** File CINIT ***


DLIMIT #DLIMIT  (This is the number of times the computer
                 will do calculations. ie. The number of
                 sample periods.)
ALIMIT #ALIMIT  (This is the number of times each analog
                 calculation will be done during each sample
                 period. Effects how accurate the analog plant
                 is modeled.)
DELAY  #DELAY   (This is the number of analog calculations done
                 before the digital to analog is availible.)
 T     #T       (This is the sample rate of the digital system.
                 You must provide it if you are going to use it
                 in your equations.)
 T1    #T1     (This is the sample rate of the analog system.
                It is usually the same as T above during design;
                however, for simulation it usually works out to
                be T divided by ALIMIT since you usually want a
                more accurate plot of the analog plant to see
                what happens between the digital sample periods.
                You must provide T1 if you use it in your equa-
                tions.)(This is effectively double rate sampling.)
```

Figure 17A
CINIT

```
* CINIT.FMT
@  2, 3 SAY "DLIMIT"
@  2,10 GET DLIMIT
@  2,24 SAY "(This is the number of times the computer"
@  3,25 SAY "will do calculations. ie. The number of"
@  4,25 SAY "sample periods.)"
@  5, 3 SAY "ALIMIT"
@  5,10 GET ALIMIT
@  5,24 SAY "(This is the number of times each analog"
@  6,25 SAY "calculation will be done during each sample"
@  7,25 SAY "period. Effects how accurate the analog plant"
@  8,25 SAY "is modeled.)"
@  9, 3 SAY "DELAY"
@  9,10 GET DELAY
@  9,24 SAY "(This is the number of analog calculations done"
@ 10,25 SAY "before the digital to analog is availible.)"
@ 11, 4 SAY "T"
@ 11,10 GET T
@ 11,24 SAY "(This is the sample rate of the digital system."
@ 12,25 SAY "You must provide it if you are going to use it"
@ 13,25 SAY "in your equations.)"
@ 14, 4 SAY "T1"
@ 14,10 GET T1
@ 14,24 SAY "(This is the sample rate of the analog system."
@ 15,25 SAY "It is usually the same as T above during design;"
@ 16,25 SAY "however, for simulation it usually works out to"
@ 17,25 SAY "be T divided by ALIMIT since you usually want a"
@ 18,25 SAY "more accurate plot of the analog plant to see"
@ 19,25 SAY "what happens between the digital sample periods."
@ 20,25 SAY "You must provide T1 if you use it in your equa-"
@ 21,25 SAY "tions.)(This is effectively double rate sampling.)"
```

Figure 17B

CINIT Format

```
* CONTROL.CMD
DO DCLMON
STORE Ø TO IND
LOCATE FOR FUNCTYPE = 'AA'
DO PROC
STORE Ø TO IND
LOCATE FOR FUNCTYPE = 'AD'
DO PROC
STORE Ø TO IND
LOCATE FOR FUNCTYPE = 'DD'
DO PROC
STORE Ø TO IND
LOCATE FOR FUNCTYPE = 'DA'
DO PROC
DO DATA
DO FORM
QUIT TO 'SUBMIT1 FORM'
SET TALK ON
RETURN
```

```
* DCLMON.CMD
SET ALTERNATE TO B:DCLMON  * B: not needed if single disk
SET ALTERNATE ON
LOCATE
DO WHILE .NOT. EOF
  ? '    '+FUNCNAME+' ext entry(fixed(7)) returns(ptr),'
  CONTINUE
ENDDO
COUNT FOR SAVEDATA TO DATA1
? '    T float static ext init('+str(T,8,4)+'),'
? '    T1 float static ext init('+str(T1,8,4)+'),'
? '    DLIMIT fixed static init('+ str(DLIMIT,4)+'),'
? '    ALIMIT float static init('+ str(ALIMIT,4)+'),'
? '    delay float static init('+ str(DELAY,4)+'),'
? '   1 dat ext,'
? '     2 data('+str(DATA1,3)+') dec(15,6)'
SET ALTERNATE OFF
STORE 'AA' TO T:NAME
DO MONDCL
STORE 'AL' TO T:NAME
DO MONDCL
STORE 'DL' TO T:NAME
LO MONDCL
STORE 'DA' TO T:NAME
DO MONDCL
RETURN
```

```
* MONDCL.CMD
SET ALTERNATE TO B:&T:NAME  * B: not needed if single disk
SET ALTERNATE ON
LOCATE FOR FUNCTYPE = T:NAME
IF EOF THEN
  ?
ELSE
   DO WHILE .NOT. EOF
     ? 'dummy_ptr = '+funcname+'(code)\'
     CONTINUE
   ENDDO
ENDIF
SET ALTERNATE OFF
RETURN
```

```
* PROC.CMD
DO WHILE .NOT. EOF
  STORE IND+1 TO IND
  STORE 'DCLFUN.'+$(FUN,IND,1)+FUNCTYPE TO T:NAME
  SET ALTERNATE TO B:&T:NAME  * B: not needed if single disk
  SET ALTERNATE ON
  ? FUNCNAME+':proc(code) returns(ptr)\'
  ? 'dcl'
  ? 'nr_inputs fixed(7) static init('+NR:INPUTS+'),'
  ? INPUT:A+' ext entry(fixed(7)) returns(ptr),'
  ? INPUT:B+' ext entry(fixed(7)) returns(ptr),'
  ? INPUT:C+' ext entry(fixed(7)) returns(ptr),'
  ? INPUT:D+' ext entry(fixed(7)) returns(ptr),'
  ? INPUT:E+' ext entry(fixed(7)) returns(ptr),'
  ? 'cod(5) fixed(7) static init('+A1P2D+','+B1P2D+;
            ','+C1P2D+','+D1P2D+','+E1P2D+'),'
  STORE STR(FK,8,4) TO TEMPOUT
  ? 'F float static init('+TEMPOUT+'),'
  STORE STR(FKD,8,4) TO TEMPOUT
  ? '0 float static init('+TEMPOUT+')\'
  ? 'dcl'
  ? '  1 dat ext,'
  ? '    2 data('+STR(data1,3)+') dec(15,6)\'
  SET ALTERNATE OFF
  STORE 'INPUT.'+$(FUN,IND,1)+FUNCTYPE TO T:NAME
  SET ALTERNATE TO B:&T:NAME  * B: not needed if single disk
  SET ALTERNATE ON
  ? 'input(1) = '+INPUT:A+'\'
  ? 'input(2) = '+INPUT:B+'\'
  ? 'input(3) = '+INPUT:C+'\'
  ? 'input(4) = '+INPUT:D+'\'
  ? 'input(5) = '+INPUT:E+'\'
  SET ALTERNATE OFF
  STORE 'FUNC.'+$(FUN,IND,1)+FUNCTYPE TO T:NAME
  SET ALTERNATE TO B:&T:NAME * B: not needed if single disk
  SET ALTERNATE ON
  ? EQUATION
  SET ALTERNATE OFF
  STORE 'OUT.'+$(FUN,IND,1)+FUNCTYPE TO T:NAME
  SET ALTERNATE TO B:&T:NAME * b: not needed if single disk
  SET ALTERNATE ON
  IF SAVEDATA
    STORE ELEMENTS+1 TO ELEMENTS
    ? 'data('+STR(ELEMENTS,3)+') = ftc('+DATANAME+')\'
    ? [put skip list('data(]+STR(ELEMENTS,3)+[) = ',]
    ?? DATANAME+')\'
  ELSE
    ?
  ENDIF
  SET ALTERNATE OFF
  CONTINUE
ENDDO
RETURN
```

```
* DATA.CMD
SET ALTERNATE TO B:DATAINFO  * B: not needed if single disk
SET ALTERNATE ON
? 'time1 fixed static init('+STR(DLIMIT*ALIMIT,4)+'),'
? 'nr_data fixed static init('+STR(DATA1,3)+'),'
COUNT FOR SAVEDATA .AND. (FUNCTYPE = 'AA') TO IND
? 'nr_aa fixed static init('+STR(IND,2)+'),'
COUNT FOR SAVEDATA .AND. (FUNCTYPE = 'AD') TO IND
? 'nr_ad fixed static init('+STR(IND,2)+'),'
COUNT FOR SAVEDATA .AND. (FUNCTYPE = 'DD') TO IND
? 'nr_dd fixed static init('+STR(IND,2)+'),'
COUNT FOR SAVEDATA .AND. (FUNCTYPE = 'DA') TO IND
? 'nr_da fixed static init('+STR(IND,2)+'),'
? 'name('+STR(DATA1,3)+') char(8) static init('
STORE 'AA' TO T:NAME
DO DATA1
LOCATE FOR FUNCTYPE = 'AA'
IF .NOT. EOF
  ?? ','
ENDIF
STORE 'AD' TO T:NAME
DO DATA1
LOCATE FOR FUNCTYPE = 'AD'
IF .NOT. EOF
  ?? ','
ENDIF
STORE 'DD' TO T:NAME
DO DATA1
LOCATE FOR FUNCTYPE = 'DD'
IF .NOT. EOF
  ?? ','
ENDIF
STORE 'DA' TO T:NAME
DO DATA1
?? ')'
SET ALTERNATE OFF
RETURN



* DATA1.CMD
LOCATE FOR SAVEDATA .AND. (FUNCTYPE = T:NAME)
DO WHILE .NOT. EOF
   ? ['']+FUNCNAME+['']
   CONTINUE
   IF .NOT. EOF
     ?? ','
   ENDIF
ENDDO
RETURN
```

```
* FORM.CMD
SET ALTERNATE TO FORM.SUB
SET ALTERNATE ON
? 'B:'     * B: not needed if single disk
? 'XSUB'
? 'REFORM AA.TXT AA.TTT'
? 'REFORM AD.TXT AD.TTT'
? 'REFORM DD.TXT DD.TTT'
? 'REFORM DA.TXT DA.TTT'
? 'PLI MONITOR'
STORE 'AA' TO T:NAME
DO DCLFUN
STORE 'AD' TO T:NAME
DO DCLFUN
STORE 'DD' TO T:NAME
DO DCLFUN
STORE 'DA' TO T:NAME
DO DCLFUN
? 'PLI PRINT'
? 'LINK PRINT'
? 'LINK ANSWER=MONITOR'
LOCATE
DO WHILE .NOT. EOF
 ?? ','+TRIM(FUNCNAME)
 CONTINUE
ENDDO
?
SET ALTERNATE OFF
RETURN


* DCLFUN.CMD
LOCATE FOR FUNCTYPE = T:NAME
STORE Ø TO IND
DO WHILE .NOT. EOF
   STORE IND+1 TO IND
   STORE $(FUN,IND,1)+FUNCTYPE TO EXT:NAME
   DO REFORM
   CONTINUE
ENDDO
RETURN


* REFORM.CMD
? 'REFORM INPUT.'+EXT:NAME+' INPUT.TTT'
? 'REFORM DCLFUN.'+EXT:NAME+' DCLFUN.TTT'
? 'PIP FUNC.TXT=FUNC.'+EXT:NAME
? 'REFORM OUT.'+EXT:NAME+' OUT.TTT'
? 'PLI '+FUNCTYPE
? 'PIP '+TRIM(FUNCNAME)+'.REL='+FUNCTYPE+'.REL'
RETURN
```

```
* SHORT.CMD

SET FORMAT TO CINIT   * See figure 17A and 17B
READ
SET TALK OFF
SET ALTERNATE TO B:DCLMON  *B: not needed if using a single disk
SET ALTERNATE ON
LOCATE
DO WHILE .NOT. EOF
  ? '    '+FUNCNAME+' ext entry(fixed(7)) returns(ptr),'
  CONTINUE
ENDDO
COUNT FOR SAVEDATA TO DATA1
? '    T float static ext init('+str(T,8,4)+'),'
? '    T1 float static ext init('+str(T1,8,4)+'),'
? '    DLIMIT fixed static init('+ str(DLIMIT,4)+'),'
? '    ALIMIT float static init('+ str(ALIMIT,4)+'),'
? '    delay float static init('+ str(DELAY,4)+'),'
? '  1 dat ext,'
? '    2 data('+str(DATA1,3)+') dec(15,6)'
SET ALTERNATE OFF
SET ALTERNATE TO SFORM.SUB
SET ALTERNATE ON
? 'XSUB'
? 'B:'          * B: not needed if using a single disk
? 'PLI MONITOR'
? 'PLI PRINT'
? 'LINK PRINT'
? 'LINK ANSWER=MONITOR'
LOCATE
DO WHILE .NOT. EOF
 ?? ','+TRIM(FUNCNAME)
 CONTINUE
ENDDO
?
SET ALTERNATE OFF
QUIT TO 'SUBMIT1 SFORM'
RETURN
```

APPENDIX B

PL/1-80 PROGRAMS

AA	PLI	:	AD	PLI	:	DA	PLI	: DCLINP TXT
DD	PLI	:	LINK	COM	:	MONITOR	PLI	: PLI COM
PLIØ	OVL	:	PLI1	OVL	:	PLI2	OVL	: PLILIB IRL
PIP	COM	:	REFORM	PLI	:	REFORM	REL	: REFORM COM
REFORM	SYM	:	DCLMON	TXT	:	AA	TXT	: AD TXT
DD	TXT	:	DA	TXT	:	DCLFUN	AAA	: INPUT AAA
FUNC	AAA	:	OUT	AAA	:	DCLFUN	BAA	: INPUT BAA
FUNC	BAA	:	OUT	BAA	:	DCLFUN	CAA	: INPUT CAA
FUNC	CAA	:	OUT	CAA	:	DCLFUN	AAD	: INPUT AAD
FUNC	AAD	:	OUT	AAD	:	DCLFUN	BAD	: INPUT BAD
FUNC	BAD	:	OUT	BAD	:	DCLFUN	ADD	: INPUT ADD
FUNC	ADD	:	OUT	ADD	:	DCLFUN	BDD	: INPUT BDD
FUNC	BDD	:	OUT	BDD	:	DCLFUN	ADA	: INPUT ADA
FUNC	ADA	:	OUT	ADA	:	DATAINFO	TXT	: AA TTT
AD	TTT	:	DD	TTT	:	DA	TTT	: MONITOR REL
INPUT	TTT	:	DCLFUN	TTT	:	OUT	TTT	: AA REL
X1N	REL	:	UN	REL	:	X2N1	REL	: AD REL
AD2	REL	:	AD1	REL	:	LD	REL	: E2K REL
EK	REL	:	DA	REL	:	FUNC	TXT	: DA1 REL
PRINT	REL	:	PRINT	COM	:	PRINT	SYM	: DATA FLT
ANSWER	COM	:	ANSWER	SYM	:	PRINT	PLI	

Figure 18

Programs Required for PL/1-80 Operation

```
MONITOR: proc options(main,stack(512));

dcl
   ftc entry (float binary) returns(char(17) var);
dcl
   TIME fixed ext static init(0),
   DCLOCK fixed ext static init(0),
   ACLOCK fixed ext static init(0),
   AD(8) float ext static init(0,0,0,0,0,0,0,0),
   DA(8) float ext static init(0,0,0,0,0,0,0,0),
   dummy_ptr ptr,
   code fixed(7),
   CF float ext,  /* correction factor for error signal */
   datafile file;
dcl
  %include 'dclmon.txt';
  ;

open file(datafile) record sequential output title('data.flt')
            env(b(512));
CF = ftc(exp(char(-delay/ALIMIT)));
do while (DCLOCK <= DLIMIT);
   PUT SKIP LIST(DCLOCK,'DCLOCK'); /* EDIT */
   code = 1;
   ACLOCK = 0;
   %include 'ad.ttt';
   %include 'dd.ttt';
   code = 1;
   do while (ACLOCK <= ALIMIT - 1);
      PUT SKIP LIST(ACLOCK,'ACLOCK',TIME,'TIME'); /* EDIT */
      if ACLOCK = delay then do;
         %include 'da.ttt';
      end; /* if */
      %include 'aa.ttt';
      write file(datafile) from(dat);
      TIME = TIME + 1;
      ACLOCK = ACLOCK + 1;
   end; /* do */
   DCLOCK = DCLOCK + 1;
end; /* do */
close file(datafile);
end MONITOR;
```

```
/* AA.PLI */
%include 'dclfun.ttt';
dcl
  ftc entry (float binary) returns(char(17) var);
dcl
   TIME fixed ext,
   T float ext,
   T1 float ext,
   input(5) variable entry(fixed(7)) returns(ptr),
   EA fixed static init(0),
   EA1 fixed static init(0),
   DA(8) float ext,
   ACLOCK fixed ext,
   code fixed(7),
   result ptr,
   m(5) label,
   (ia_ptr,ib_ptr,ic_ptr,id_ptr,ie_ptr) ptr;
%include 'dclinp.txt';

%include 'input.ttt';
if code = 1 then do;
   if EA < TIME then do;
     if (nr_inputs> 0) & (nr_inputs< 6)
       then do;
         m(1)=la;
         m(2)=lb;
         m(3)=lc;
         m(4)=ld;
         m(5)=le;
         goto m(nr_inputs);
       end; /* then */
       else goto lf;
     le: ie_ptr = input (5) (cod(5));
     ld: id_ptr = input (4) (cod(4));
     lc: ic_ptr = input (3) (cod(3));
     lb: ib_ptr = input (2) (cod(2));
     la: ia_ptr = input (1) (cod(1));
     lf: 0 = F;
     EA1 = TIME;
     %include 'func.txt';
     ;
     EA = TIME;
   end; /* if EA */
end; /* if code then */
else
   if EA1 < TIME then do;
     0 = F;
     EA1 = TIME;
   end; /* EA1 */
%include 'out.ttt';
result = addr(F);
return (result);
end;
```

```
/* AD.PLI */
%include 'dclfun.ttt';

dcl
  ftc entry (float binary) returns(char(17) var);
dcl
   EA fixed static init(0),
   T float ext,
   T1 float ext,
   input(5) variable entry(fixed(7)) returns(ptr),
   CF float ext,
   AD(8) float ext,
   code fixed(7),
   result ptr,
   m(5) label,
   (ia_ptr,ib_ptr,ic_ptr,id_ptr,ie_ptr) ptr;

%include 'dclinp.txt';
%include 'input.ttt';

if EA > 0 then do;
   if (nr_inputs> 0) & (nr_inputs< 6)
     then do;
       m(1)=la;
       m(2)=lb;
       m(3)=lc;
       m(4)=ld;
       m(5)=le;
       goto m(nr_inputs);
     end; /* then */
     else goto lf;
       le: ie_ptr = input (5) (cod(5));
       ld: id_ptr = input (4) (cod(4));
       lc: ic_ptr = input (3) (cod(3));
       lb: ib_ptr = input (2) (cod(2));
       la: ia_ptr = input (1) (cod(1));
       lf:
         %include 'func.txt';  /* AD(X) = IA + ETC */
         ;
end; /* then */
result = addr(F);
EA = 1;

%include 'out.ttt';
;
return (result);
end;
```

```
/* DD.PLI */
%include 'dclfun.ttt';
dcl
   ftc entry (float binary) returns(char(17) var);
dcl
   TIME fixed ext,
   T float ext,
   T1 float ext,
   input(5) variable entry(fixed(7)) returns(ptr),
   ED fixed static init(0),
   ED1 fixed static init(0),
   CF float ext,
   AD(8) float ext,
   DCLOCK fixed ext,
   code fixed(7),
   result ptr,
   m(5) label,
   (ia_ptr,ib_ptr,ic_ptr,id_ptr,ie_ptr) ptr;
%include 'dclinp.txt';

%include 'input.ttt';
if code = 1 then do;
   if ED < DCLOCK then do;
     if (nr_inputs> 0) & (nr_inputs< 6)
       then do;
         m(1)=la;
         m(2)=lb;
         m(3)=lc;
         m(4)=ld;
         m(5)=le;
         goto m(nr_inputs);
       end; /* then */
       else goto lf;
     le: ie_ptr = input (5) (cod(5));
     ld: id_ptr = input (4) (cod(4));
     lc: ic_ptr = input (3) (cod(3));
     lb: ib_ptr = input (2) (cod(2));
     la: ia_ptr = input (1) (cod(1));
     lf: 0 = F;
     ED1 = DCLOCK;
     %include 'func.txt';
     ;
     ED = DCLOCK;
   end; /* if ED */
end; /* if code then */
else do;
   if ED1 < DCLOCK then do;
     0 = F;
     ED1 = DCLOCK;
   end; /* ED1 */
end; /* else */
%include 'out.ttt';
result = addr(F);
return (result);
end;
```

```
/* DA.PLI */
%include 'dclfun.ttt';

dcl
   ftc entry (float binary) returns(char(17) var);
dcl
   TIME fixed ext,
   T float ext,
   T1 float ext,
   input(5) variable entry(fixed(7)) returns(ptr),
   CF float ext,
   AD(8) float ext,
   DA(8) float ext,
   code fixed(7),
   result ptr,
   m(5) label,
   (ia_ptr,ib_ptr,ic_ptr,id_ptr,ie_ptr) ptr;

%include 'dclinp.txt';
%include 'input.ttt';

if TIME > 0 then do;
  if (nr_inputs> 0) & (nr_inputs< 6)
    then do;
      m(1)=la;
      m(2)=lb;
      m(3)=lc;
      m(4)=ld;
      m(5)=le;
      goto m(nr_inputs);
    end; /* then */
    else goto lf;
      le: ie_ptr = input (5) (1);
      ld: id_ptr = input (4) (1);
      lc: ic_ptr = input (3) (1);
      lb: ib_ptr = input (2) (1);
      la: ia_ptr = input (1) (1);
      lf:
        %include 'func.txt';  /* DA(X) = IB - ECT */
        ;
end; /* then */
result = addr(F);

%include 'out.ttt';
;
return (result);
end;
```

```
REFORM: proc options(main);

dcl
   (input,output) file,
   buff char(1) varying;

open file(input) stream env(b(1024)) title('$1.$1');
open file(output) stream output env(b(1024)) title('$2.$2');

  do while ('1'b);
    read file (input) into (buff);
    buff = translate(buff,';',',\');
    write file (output) from (buff);
  end;
end reform;




PRINT: proc options(main);

dcl
  data dec(15,6),
  (j,k,m) fixed bin(7),
  (time,dat,temp_str) char(128) var,
  (info,input,output) file,
  %include 'datainfo.txt';
  ;

open file(input) record sequential title('DATA.FLT')
      env(b(512));
open file(output) stream output title('$1.$1')
      env(b(512));

do k = 0 to time1;
  time = ' ';
  temp_str='TIME='||char(k);
  do m = 1 to length(temp_str);
    if substr(temp_str,m,1) ^= ' ' then
      time = time||substr(temp_str,m,1);
  end;
  time = '^j^m                '||time;
  write file(output) from(time);
  do j= 1 to nr_data;
    read file(input) into(data);
    dat = '^m^j                 '||name(j)||'='||char(data);
    write file(output) from(dat);
  end;
end; /* do k */
end;
```

APPENDIX C

GRAPHS OF EXAMPLE PROBLEM RESULTS

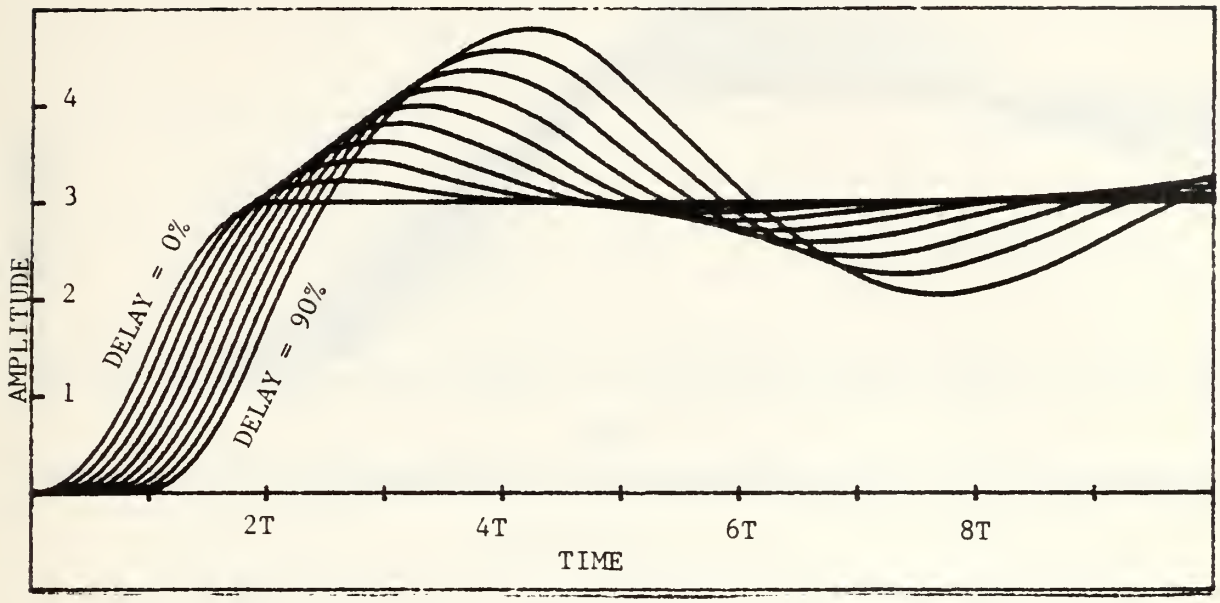


Figure 19A

Response to Unit Step Input With No Correction

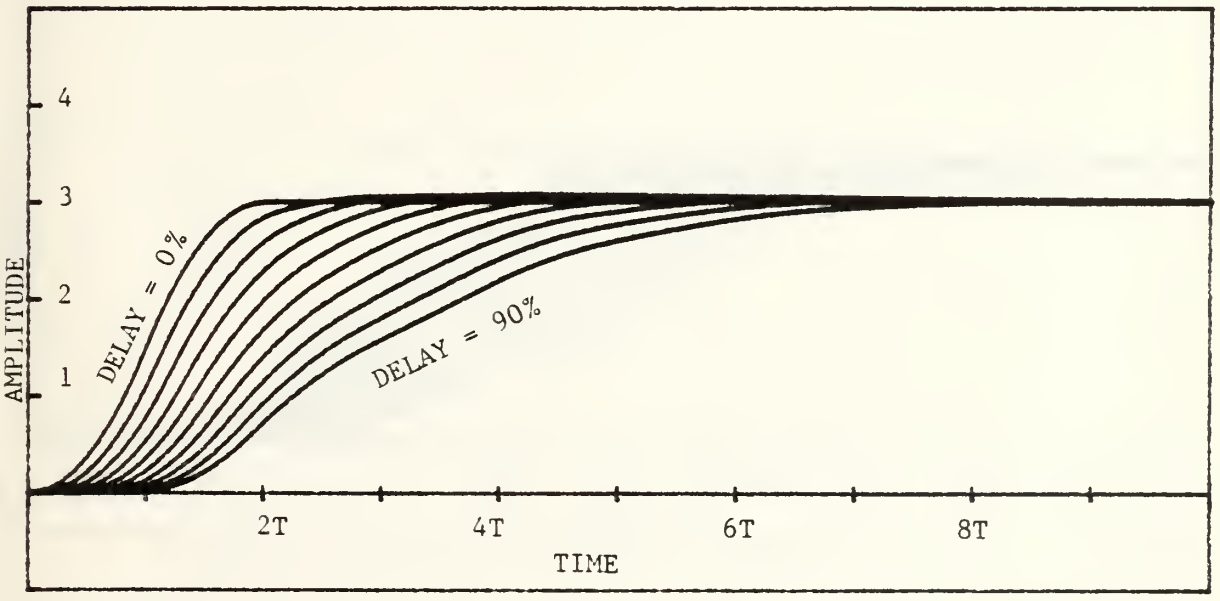


Figure 19B

Response to Unit Step Input With Correction

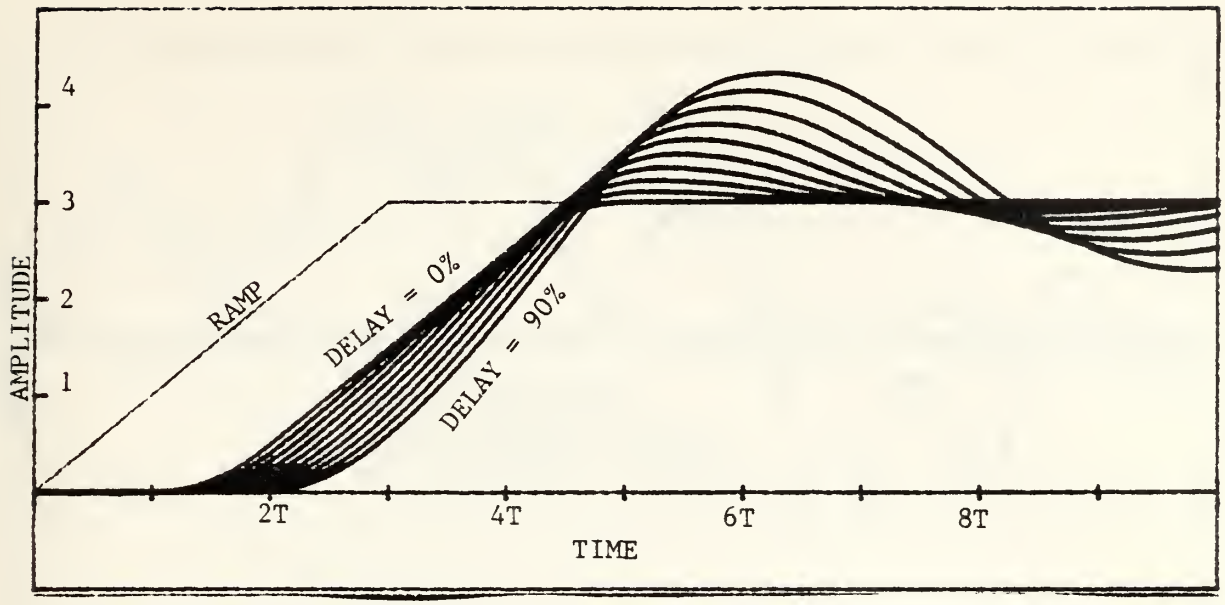


Figure 19C

Terminated Ramp Input With No Corrections

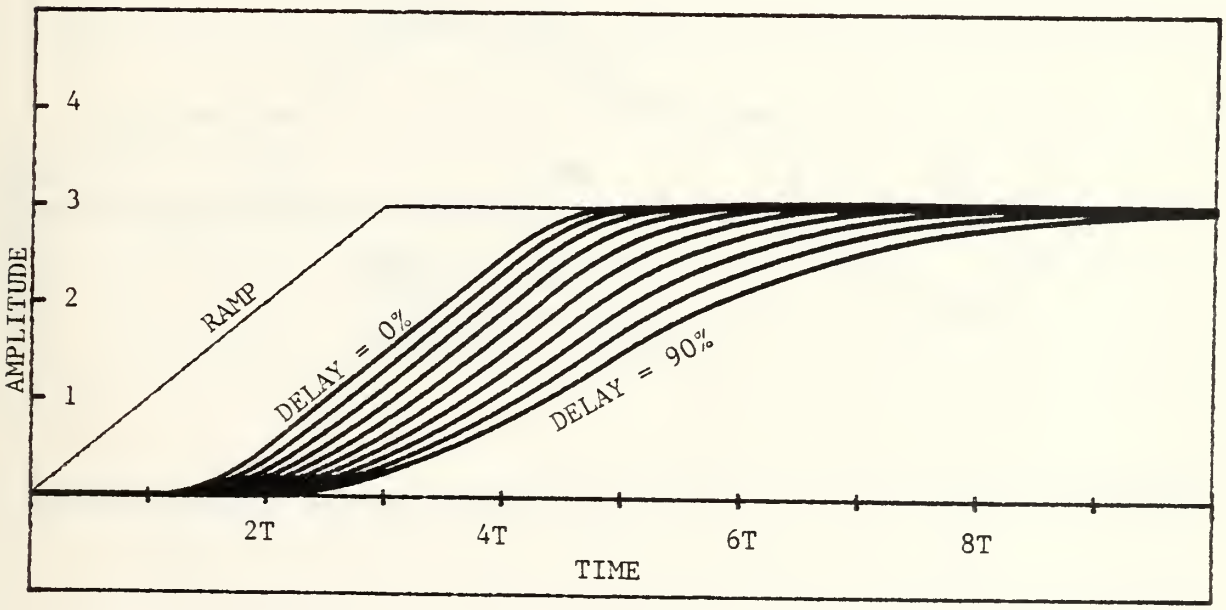


Figure 19D

Terminated Ramp Input With Corrections

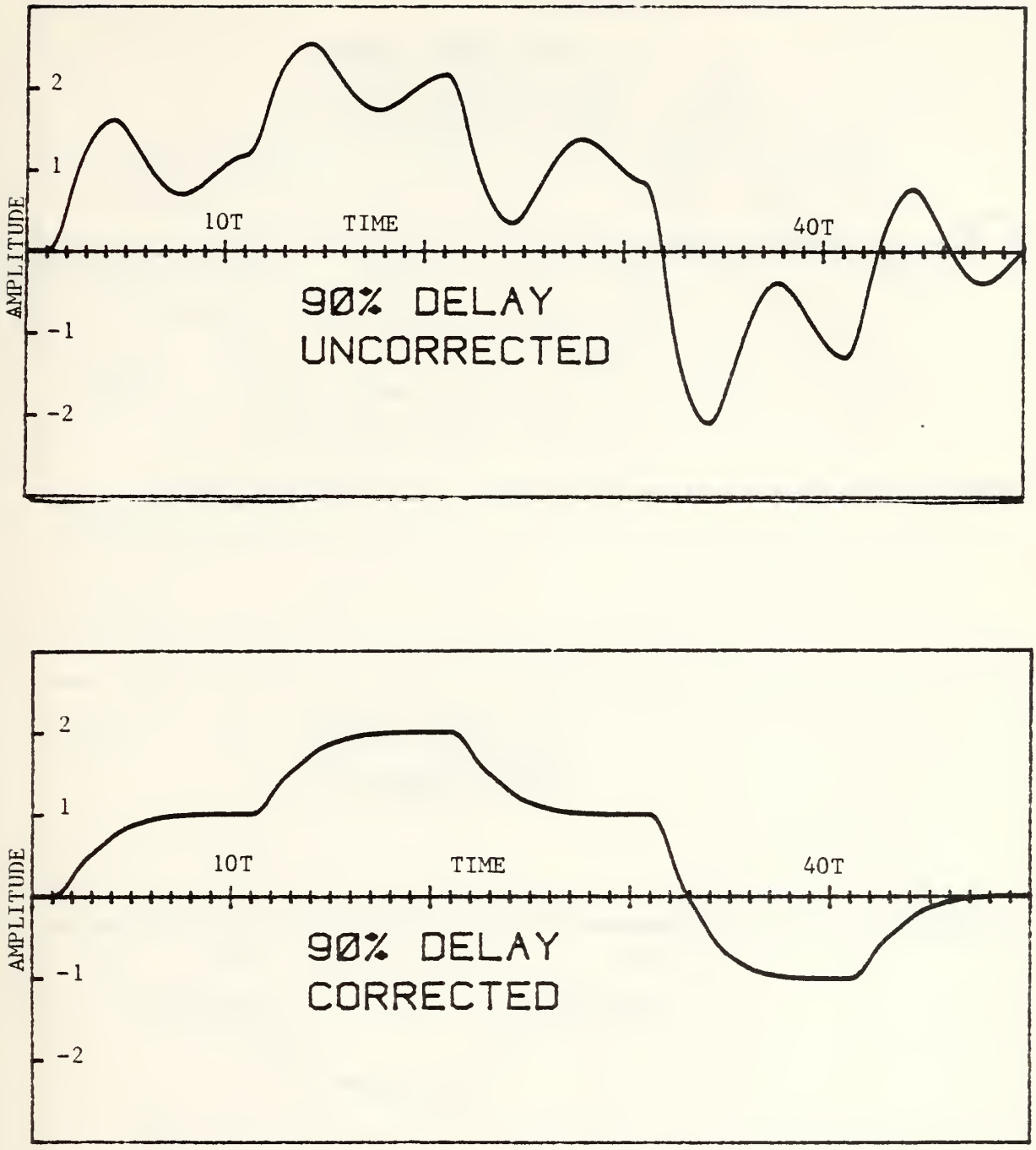
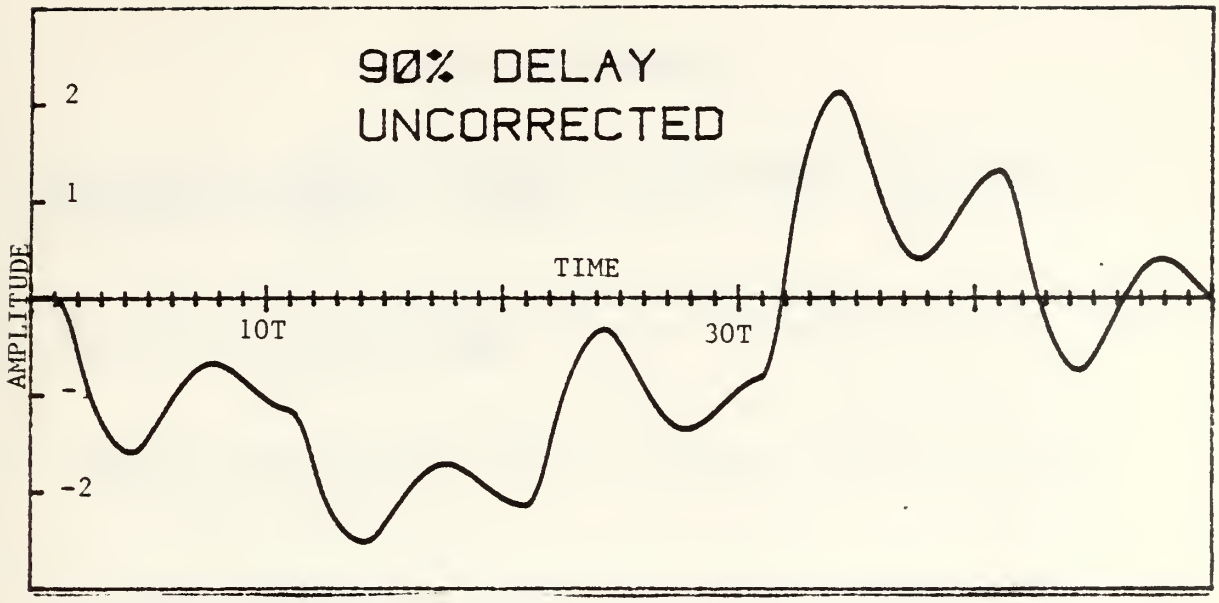


Figure 19E

Response to Variable Step Inputs with and Without Corrections



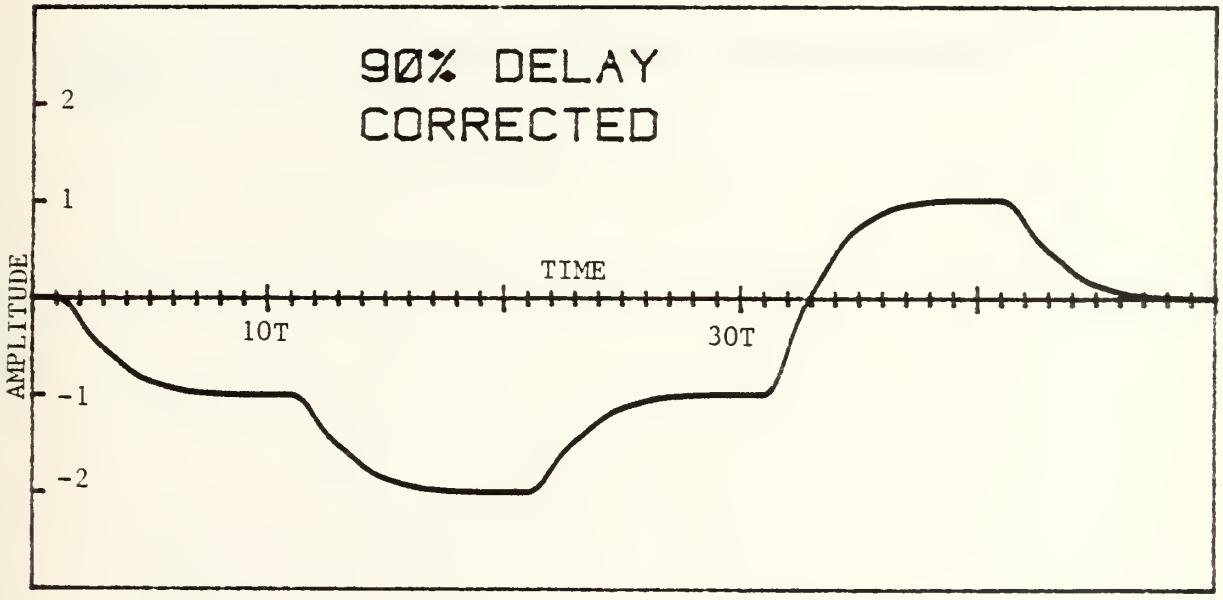


Figure 19F

Response to Variable Step Inputs with and Without Corrections

LIST OF REFERENCES

1. Ratliff,W., DBASE II Assembly - Language Relational Database Management System, Ashton Tate, Culver City, California, 1982

2. PL/1-80 Reference Manual, Digital Research, Pacific Grove, California, 1980

3. LINK-80 Users Guide, Digital Research, Pacific Grove, Califronia, 1980

4. PL/1-80 Applications Guide, Digital Research, Pacific Grove, California, 1980

5. Cadzow, J.A. and Martens, H.R., Discrete-Time and Computer Control Systems, Prentice Hall, 1970

6. Katz, P., Digital Control Using Microprocessors, Prentice Hall, 1981

BIBLIOGRAPHY

Franklin, G.F. and Powell J.I., Digital Control of Dynamic Systems, Addison Wesley, 1980

Kirk, D.E., Optimal Control Theory, Prentice Hall, 1970

Ogata, K., Modern Control Engineering, Prentice Hall, 1970

Oppenheim, A.V. and Schafer, R.W., Digital Signal Processing, Prentice Hall, 1975

Zaks, R., The CP/M Handbook With MP/M, Sybex Inc., 1980

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code Ø142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	Professor Alex Gerba, Jr., Code 62Gz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	2
5.	Professor George J. Thaler, Code 62Tr Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
6.	Professor Mitchell L. Cotton, Code 62Cc Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
7.	Professor Harold A. Titus, Code 62Ts Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
8.	Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
9.	Lt. Roger F. Johnsrud, USN % Cindy F. Lee 8961 Universe Avenue Westminster, California 92683	1